Felix Douven, Yosha op het Veld, Kasper Walraven

# Initialization

load all needed libraries and functions, check the previous tutorial how to correctly load keras and other modules

In [17]:
```
import matplotlib.pyplot as plt
import numpy as np

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D

import pandas as pd
import matplotlib.pyplot as plt

import os
import cv2
from tqdm import tqdm
```

importing the libraries used in the code.

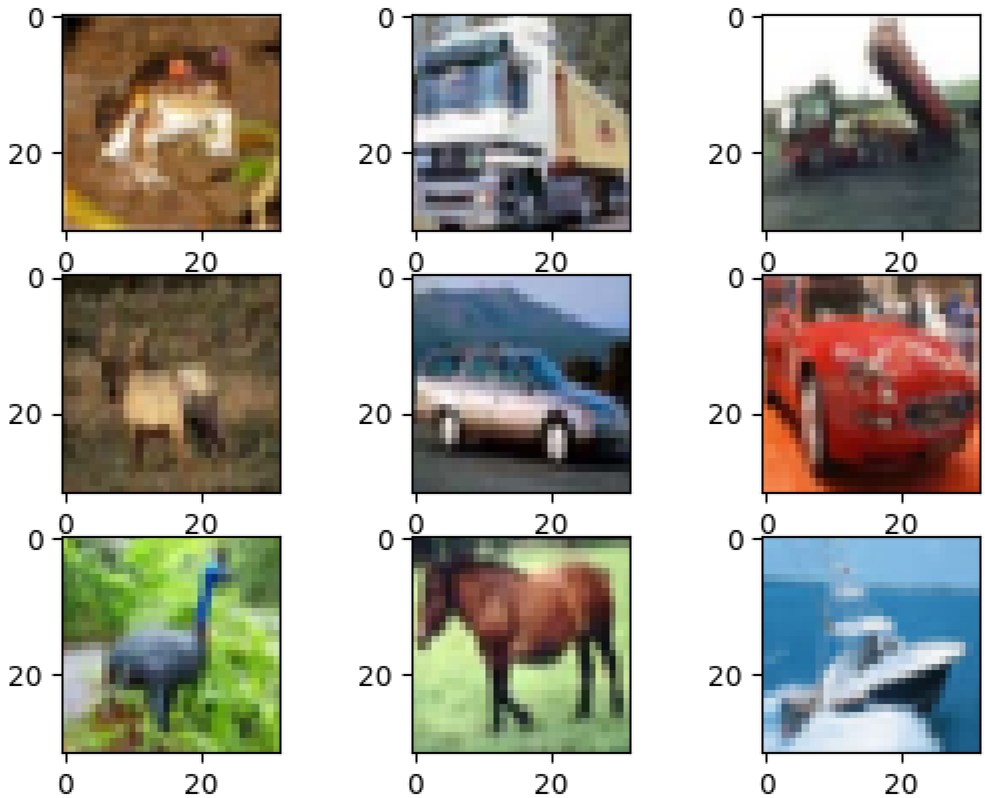
# Load dataset & Plot a subset

load your dataset and show a plot of the subset of your data

> Just remember that you must use at least 3 classes and at most 10 classes, so, in the case of the cifar10, if you decide to use 5 classes, then get rid of the other 5 to save space. In other words, choose a dataset, check the images (amount, size in pixels) and implement the steps needed shown in the provided notebook.

In [18]:
```
from matplotlib import pyplot
from keras.datasets import cifar10

(trainX, trainY), (testX, testY) = cifar10.load_data()

print('Train: X=%s, y=%s' % (trainX.shape, trainY.shape))
print('Test: X=%s, y=%s' % (testX.shape, testY.shape))
catagories = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","t
for i in range(9):
    pyplot.subplot(330 + 1 + i)
    pyplot.imshow(trainX[i])

pyplot.show()
```

```
Train: X=(50000, 32, 32, 3), y=(50000, 1)
Test: X=(10000, 32, 32, 3), y=(10000, 1)
```




importing the images and plotting a few samples.

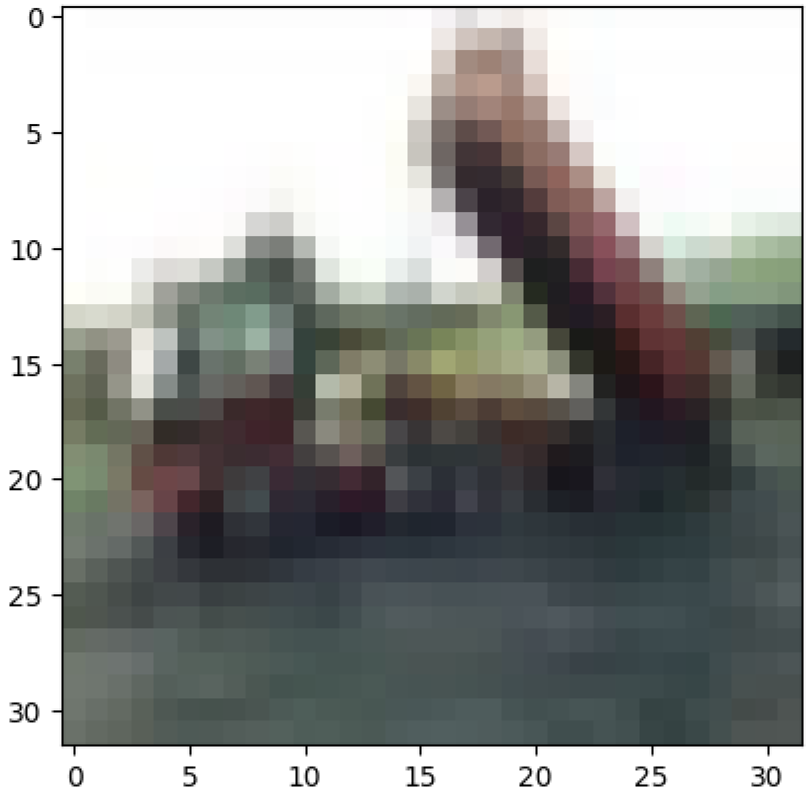
# Prepare Pixel Data

pre-process your raw input data... rescale... normalize....

In [19]:
```
t = 0
from sklearn.utils import shuffle



print("x = ", trainX[2])
plt.imshow(trainX[2], cmap=plt.cm.binary)
plt.show()

trainX = trainX.astype('float32')
testX = testX.astype('float32')
X_train = tf.keras.utils.normalize(trainX, axis=1)
X_test = tf.keras.utils.normalize(testX, axis=1)

print("na norm", X_train[2])
print("dit is een", catagories[int(trainY[2])])
plt.imshow(X_train[2], cmap=plt.cm.binary)
plt.show()

X_train, trainY = shuffle(X_train, trainY, random_state=0)
```

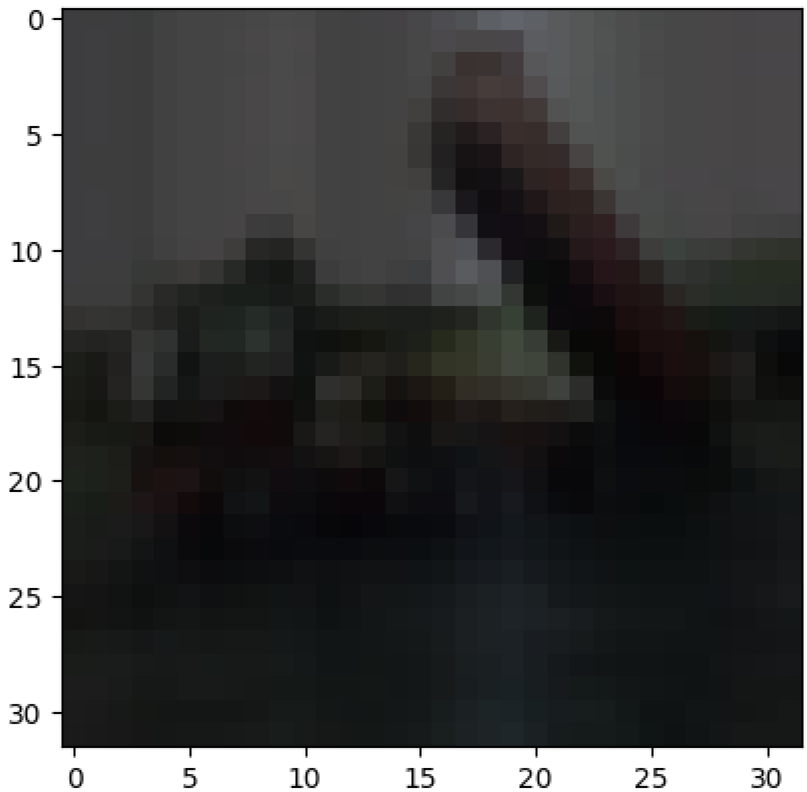
```
x =  [[[255 255 255]
  [253 253 253]
  [253 253 253]
  ...
  [253 253 253]
  [253 253 253]
  [253 253 253]]
```

```
[[255 255 255]
 [255 255 255]
 [255 255 255]
 ...
 [255 255 255]
 [255 255 255]
 [255 255 255]]

[[255 255 255]
 [254 254 254]
 [254 254 254]
 ...
 [254 254 254]
 [254 254 254]
 [254 254 254]]

...

[[113 120 112]
 [111 118 111]
 [105 112 106]
 ...
 [ 72  81  80]
 [ 72  80  79]
 [ 72  80  79]]

[[111 118 110]
 [104 111 104]
 [ 99 106  98]
 ...
 [ 68  75  73]
 [ 70  76  75]
 [ 78  84  82]]

[[106 113 105]
 [ 99 106  98]
 [ 95 102  94]
 ...
 [ 78  85  83]
 [ 79  85  83]
 [ 80  86  84]]]
```




```
na norm [[[0.24103695 0.23701033 0.2449874 ]
  [0.2431178  0.23965265 0.24643646]
  [0.24044508 0.23909946 0.24423422]
  ...
  [0.28057057 0.27200606 0.27913165]
  [0.28430322 0.27541536 0.28258422]
  [0.28472024 0.2759236  0.28289214]]

 [[0.24103695 0.23701033 0.2449874 ]
  [0.24503967 0.24154714 0.24838458]
  [0.24234582 0.24098957 0.24616493]
  ...
  [0.28278852 0.2741563  0.2813382 ]
  [0.28655067 0.27759254 0.28481808]
  [0.286971   0.27810484 0.28512844]]

 [[0.24103695 0.23701033 0.2449874 ]
  [0.24407873 0.24059989 0.24741052]
  [0.24139546 0.2400445  0.24519958]
  ...
  [0.28167954 0.27308118 0.28023493]
  [0.28542694 0.27650395 0.28370115]
  [0.2858456  0.27701423 0.2840103 ]]

 ...

 [[0.10681245 0.11153428 0.10760231]
  [0.10666433 0.11177475 0.10812034]
  [0.09978946 0.1058464  0.10232738]
  ...
  [0.07984617 0.08708494 0.08826297]
  [0.08090843 0.08708786 0.08823776]
  [0.08102711 0.08724858 0.0883339 ]]

 [[0.10492197 0.10967537 0.10568084]
  [0.09993774 0.10514405 0.10130195]
  [0.09408721 0.10017606 0.09460456]
  ...
  [0.07541028 0.08063421 0.08053996]
```

```
  [0.07866097 0.08273347 0.08377002]
  [0.08777937 0.09161101 0.09168836]]

 [[0.10019575 0.10502811 0.10087717]
  [0.09513305 0.10040783 0.09545761]
  [0.0902857  0.09639583 0.09074315]
  ...
  [0.08650002 0.09138543 0.09157283]
  [0.08877452 0.09253085 0.09270549]
  [0.09003012 0.09379222 0.09392466]]]
dit is een truck
```




the pixel data and a plot of the pictures is shown. all values scaled between 0 and 1. thus the highest value is scaled down to 1. This is why the second image is a different color than before

# Define your Model

This is the crucial part of the assignment!

We do not expect that you can/should develop your own network model, so you can take the suggested model as decribed on the given website.....but

**NOTE:**

If you run into memory and processing limitations you can reduce the amount of convolutions and dense layers, you can reduce the amount of classes, you can reduce the amount of input images, or the input images size. With a scaled down network the accuracy will be lower then with a more complex network.

- How is your model constructed, how many trainable parameters does it have, and where are they located?

In [4]:
```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Input(shape=(32, 32, 3)))
model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu'))
```

```
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928     
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 2, 2, 64)         0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 256)               0         
                                                                 
 dense (Dense)               (None, 128)               32896     
                                                                 
 dense_1 (Dense)             (None, 128)               16512     
                                                                 
 dense_2 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 107,018
Trainable params: 107,018
Non-trainable params: 0
_________________________________________________________________
```

the model has 11 layers, 9 of which are hidden. the input layer has a shape of 32*32*3. the output has 10 neurons, 1 for each class. the hidden layers consist of 3 convolution layers, each of them followed by a pooling layer. the output of these layers are run trough 2 fully connected layers of 128 neurons each.

# Fit the Model

Fitting the model is the time consuming part, this depend on the complexity of the model and the amount of training data. In the fitting process the model is first build up in memory with all the tunable parameters and intercomnnects (with random start values). This is also the limitation of some systems, all these parameters are stored in memory (or when not fitting in a swap file)

**TIP:** do not start the first time with training a lot of epochs, first see if this and all following steps in your system work and when you are sure that all works train your final model.

- Which batch size and how many epochs give a good result?

In [5]:
```
model.compile(optimizer='SGD',  # Good default optimizer to start with
              loss='sparse_categorical_crossentropy',  # how will we calculate our "erro
              metrics=['accuracy'])  # what to track
#model.fit(X_train, trainY, epochs=10, batch_size=64, validation_data=(X_test, testY), v
history = model.fit(X_train, trainY, epochs=100)  # train the model
```

```
Epoch 1/100
1563/1563 [==============================] - 15s 6ms/step - loss: 2.3005 - accuracy: 0.1
247
Epoch 2/100
1563/1563 [==============================] - 8s 5ms/step - loss: 2.2799 - accuracy: 0.17
88
Epoch 3/100
1563/1563 [==============================] - 8s 5ms/step - loss: 2.1250 - accuracy: 0.21
99
Epoch 4/100
1563/1563 [==============================] - 8s 5ms/step - loss: 1.9381 - accuracy: 0.29
77
Epoch 5/100
1563/1563 [==============================] - 8s 5ms/step - loss: 1.7964 - accuracy: 0.35
06
Epoch 6/100
1563/1563 [==============================] - 9s 6ms/step - loss: 1.7068 - accuracy: 0.38
40
Epoch 7/100
1563/1563 [==============================] - 8s 5ms/step - loss: 1.6286 - accuracy: 0.41
63
Epoch 8/100
1563/1563 [==============================] - 9s 6ms/step - loss: 1.5626 - accuracy: 0.44
20
Epoch 9/100
1563/1563 [==============================] - 9s 6ms/step - loss: 1.5037 - accuracy: 0.46
18
Epoch 10/100
1563/1563 [==============================] - 12s 8ms/step - loss: 1.4545 - accuracy: 0.4
842
Epoch 11/100
1563/1563 [==============================] - 9s 6ms/step - loss: 1.4060 - accuracy: 0.50
34
Epoch 12/100
1563/1563 [==============================] - 8s 5ms/step - loss: 1.3554 - accuracy: 0.51
70
Epoch 13/100
1563/1563 [==============================] - 9s 6ms/step - loss: 1.3139 - accuracy: 0.53
52
Epoch 14/100
1563/1563 [==============================] - 9s 6ms/step - loss: 1.2731 - accuracy: 0.55
01
Epoch 15/100
1563/1563 [==============================] - 9s 6ms/step - loss: 1.2288 - accuracy: 0.56
66
Epoch 16/100
1563/1563 [==============================] - 9s 6ms/step - loss: 1.1926 - accuracy: 0.58
04
Epoch 17/100
1563/1563 [==============================] - 9s 6ms/step - loss: 1.1557 - accuracy: 0.59
48
Epoch 18/100
1563/1563 [==============================] - 9s 6ms/step - loss: 1.1215 - accuracy: 0.60
50
Epoch 19/100
1563/1563 [==============================] - 9s 6ms/step - loss: 1.0868 - accuracy: 0.62
09
```

```
Epoch 20/100
1563/1563 [==============================] - 8s 5ms/step - loss: 1.0532 - accuracy: 0.63
02
Epoch 21/100
1563/1563 [==============================] - 9s 6ms/step - loss: 1.0253 - accuracy: 0.64
33
Epoch 22/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.9986 - accuracy: 0.65
26
Epoch 23/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.9668 - accuracy: 0.66
27
Epoch 24/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.9418 - accuracy: 0.67
06
Epoch 25/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.9215 - accuracy: 0.67
78
Epoch 26/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.8973 - accuracy: 0.68
59
Epoch 27/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.8736 - accuracy: 0.69
37
Epoch 28/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8511 - accuracy: 0.70
30
Epoch 29/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.8296 - accuracy: 0.70
91
Epoch 30/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.8113 - accuracy: 0.71
70
Epoch 31/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.7908 - accuracy: 0.72
40
Epoch 32/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.7723 - accuracy: 0.72
88
Epoch 33/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.7576 - accuracy: 0.73
61
Epoch 34/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7380 - accuracy: 0.74
14
Epoch 35/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.7205 - accuracy: 0.74
80
Epoch 36/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7044 - accuracy: 0.75
39
Epoch 37/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.6866 - accuracy: 0.75
84
Epoch 38/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.6700 - accuracy: 0.76
59
Epoch 39/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.6539 - accuracy: 0.77
34
Epoch 40/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.6365 - accuracy: 0.77
71
Epoch 41/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.6237 - accuracy: 0.78
13
```

```
Epoch 42/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.6080 - accuracy: 0.78
66
Epoch 43/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.5941 - accuracy: 0.79
04
Epoch 44/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.5766 - accuracy: 0.79
76
Epoch 45/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.5645 - accuracy: 0.80
18
Epoch 46/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.5502 - accuracy: 0.80
74
Epoch 47/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.5357 - accuracy: 0.81
22
Epoch 48/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.5214 - accuracy: 0.81
50
Epoch 49/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.5078 - accuracy: 0.82
13
Epoch 50/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4944 - accuracy: 0.82
58
Epoch 51/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.4822 - accuracy: 0.83
02
Epoch 52/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.4690 - accuracy: 0.83
37
Epoch 53/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.4603 - accuracy: 0.83
80
Epoch 54/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4440 - accuracy: 0.84
45
Epoch 55/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.4301 - accuracy: 0.84
82
Epoch 56/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.4192 - accuracy: 0.85
24
Epoch 57/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.4059 - accuracy: 0.85
39
Epoch 58/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.3960 - accuracy: 0.85
95
Epoch 59/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.3845 - accuracy: 0.86
25
Epoch 60/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.3738 - accuracy: 0.86
56
Epoch 61/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.3577 - accuracy: 0.87
37
Epoch 62/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.3484 - accuracy: 0.87
72
Epoch 63/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.3398 - accuracy: 0.87
95
```

```
Epoch 64/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.3276 - accuracy: 0.88
32
Epoch 65/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.3172 - accuracy: 0.88
77
Epoch 66/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.3087 - accuracy: 0.88
90
Epoch 67/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.2943 - accuracy: 0.89
66
Epoch 68/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.2887 - accuracy: 0.89
84
Epoch 69/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2787 - accuracy: 0.90
02
Epoch 70/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.2668 - accuracy: 0.90
55
Epoch 71/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2569 - accuracy: 0.90
79
Epoch 72/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.2544 - accuracy: 0.91
01
Epoch 73/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.2433 - accuracy: 0.91
46
Epoch 74/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2343 - accuracy: 0.91
80
Epoch 75/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.2292 - accuracy: 0.91
92
Epoch 76/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2199 - accuracy: 0.92
25
Epoch 77/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.2098 - accuracy: 0.92
41
Epoch 78/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.2051 - accuracy: 0.92
72
Epoch 79/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.1929 - accuracy: 0.93
20
Epoch 80/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.1908 - accuracy: 0.93
39
Epoch 81/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.1812 - accuracy: 0.93
49
Epoch 82/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.1733 - accuracy: 0.94
06
Epoch 83/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.1717 - accuracy: 0.93
93
Epoch 84/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.1673 - accuracy: 0.94
18
Epoch 85/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1563 - accuracy: 0.94
50
```

```
Epoch 86/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1454 - accuracy: 0.94
95
Epoch 87/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1514 - accuracy: 0.94
73
Epoch 88/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1473 - accuracy: 0.94
84
Epoch 89/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1419 - accuracy: 0.94
99
Epoch 90/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1342 - accuracy: 0.95
27
Epoch 91/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1287 - accuracy: 0.95
47
Epoch 92/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1193 - accuracy: 0.95
90
Epoch 93/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.1067 - accuracy: 0.96
35
Epoch 94/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.1123 - accuracy: 0.96
18
Epoch 95/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.1155 - accuracy: 0.96
03
Epoch 96/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.1091 - accuracy: 0.96
22
Epoch 97/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.1044 - accuracy: 0.96
37
Epoch 98/100
1563/1563 [==============================] - 9s 5ms/step - loss: 0.1202 - accuracy: 0.95
72
Epoch 99/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.1125 - accuracy: 0.96
00
Epoch 100/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.1042 - accuracy: 0.96
29
```

Here the optimiser is chosen and model is trained with 100 iterations.

# Evaluate Model

Show the model accuracy after the training process ...

- How accurate is your final model?

In [6]:
```
val_loss, val_acc = model.evaluate(X_test, testY)  # evaluate the out of sample data wit
print(val_loss)  # model's loss (error)
print(val_acc)  # model's accuracy
```

```
313/313 [==============================] - 1s 4ms/step - loss: 2.4992 - accuracy: 0.6490
2.4991886615753174
0.6489999890327454
```

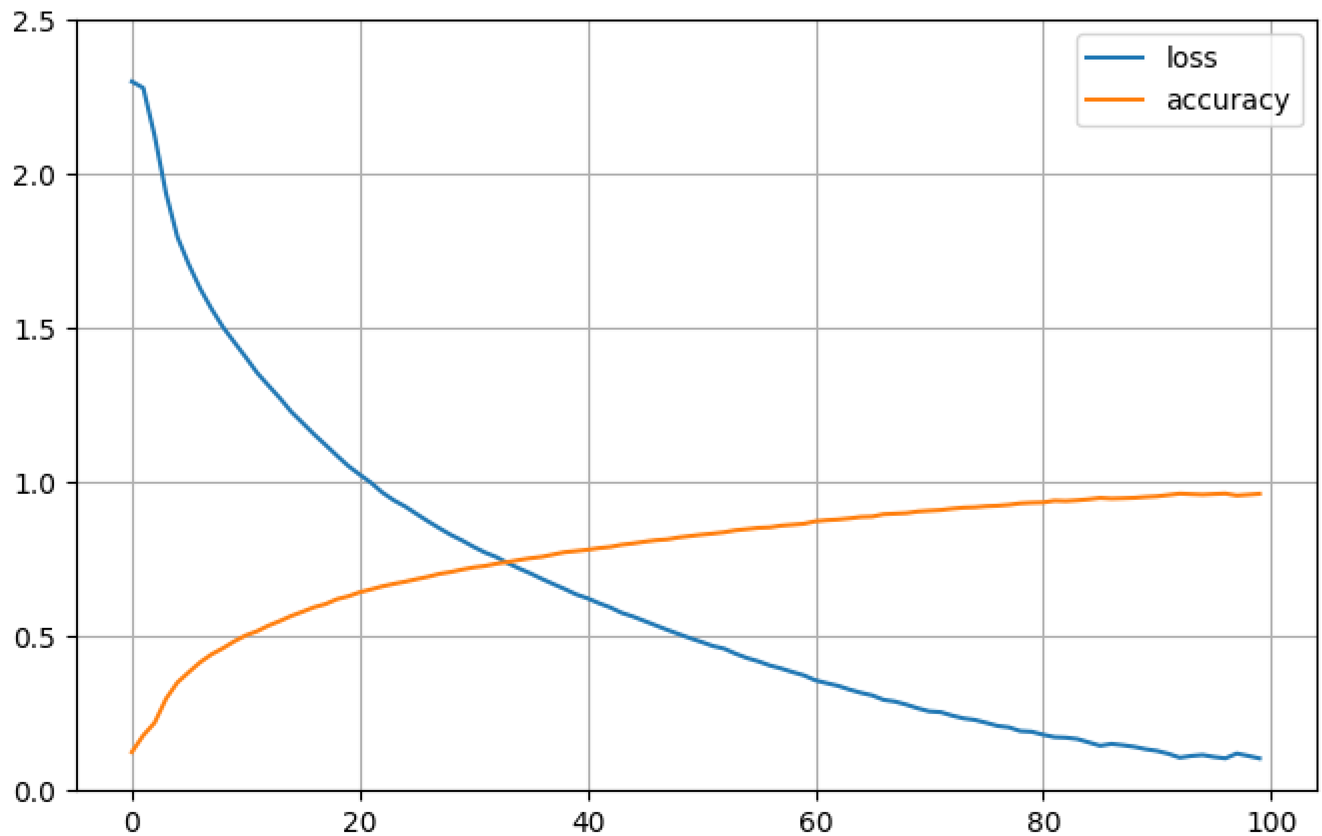
Here the model is tested with the given test data. The accuracy is shown

# learning curves

Show the learning curves of your training sequence, of accuracy, value_accuracy and loss, value_loss

- Explain what the difference is between the therms accuracy and value_accuracy? (what do they represent)

In [7]:
```
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 2.5)
plt.show()
```




the loss and accurcy during training is plotted. the graph flattens off at the end near 96%

# Save model

Save the model for later usage

In [8]:
```
model.save('Ass2B')
new_model = tf.keras.models.load_model('Ass2B')
```

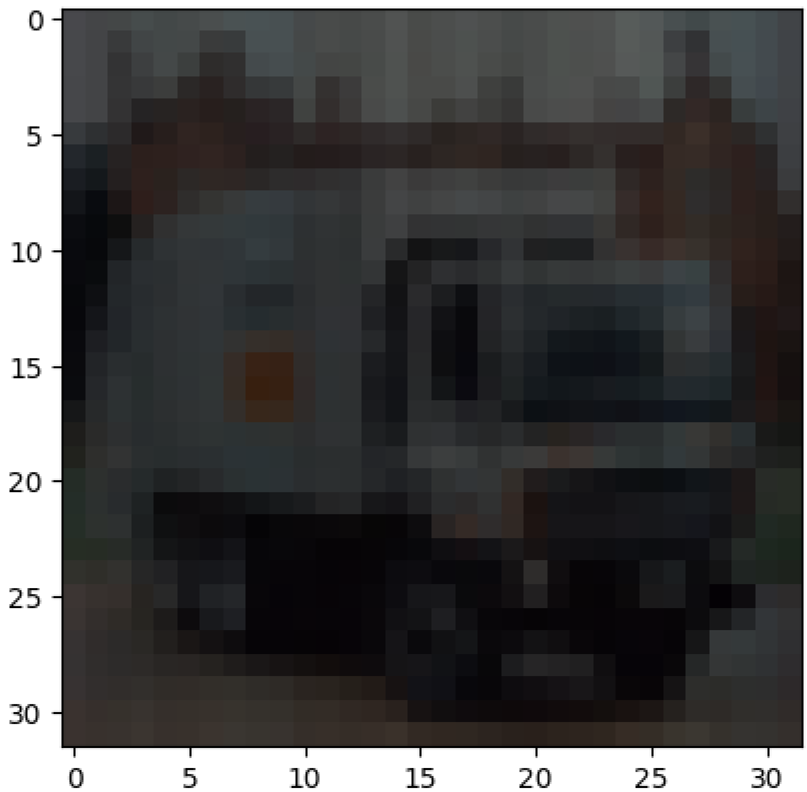
```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compile
d_convolution_op, _jit_compiled_convolution_op while saving (showing 3 of 3). These func
tions will not be directly callable after loading.
INFO:tensorflow:Assets written to: Ass2B\assets
INFO:tensorflow:Assets written to: Ass2B\assets
```

# Evaluate Final Model

After training and saving the model you can deploy this model on any given input image. You can start a new application in where you import this model and apply it on any given imput images, so you can just load the model and don't need the timeconsuming training anymore.

In [9]:
```
predictions = new_model.predict([X_test])

plt.imshow(X_test[11], cmap=plt.cm.binary)
plt.show()

print("prediction =", catagories[np.argmax(predictions[11])])
```

```
313/313 [==============================] - 1s 3ms/step
```




```
prediction = truck
```

Here we predict one image to double check the model. this image is out of the test data. The model classifies the image as truck. as seen by the image this is correct.

## Make Prediction

We can use our saved model to make a prediction on new images that are not trained on... make sure the input images receive the same pre-processing as the images you trained on.

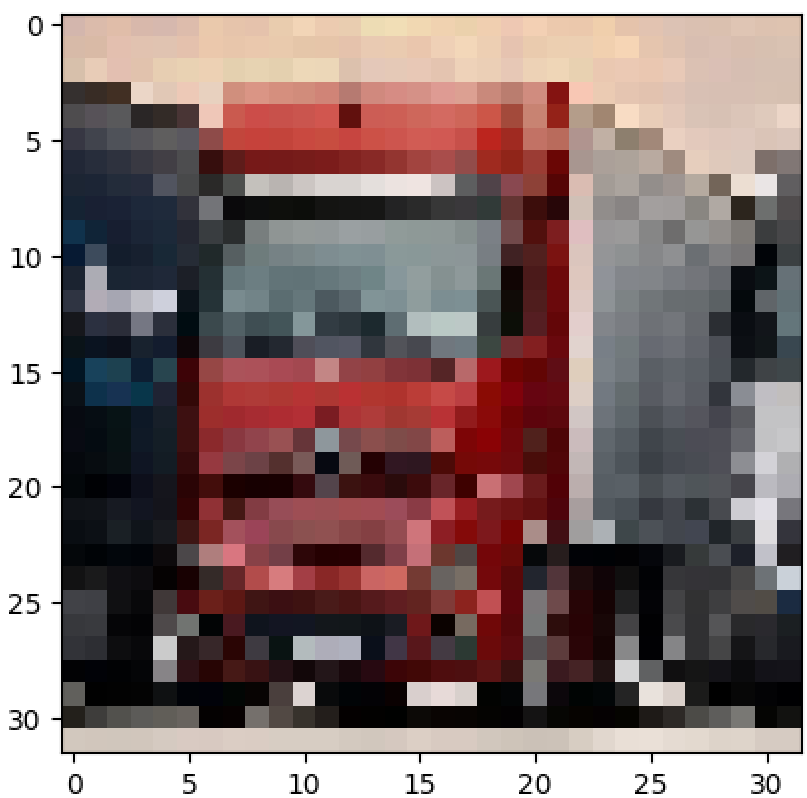
So fetch some images from the internet (similar classes, but not from your dataset), prepare them to fit your network and classify them. Do this for **10 images per class** and show the results!

- How good is the detection on you real dataset? (show some statistics)

In [11]:
```
#yosha
#DATADIR = "C:/Users/yosha19/OneDrive - Office 365 Fontys/General/Ass2B/plaatjes"

#kasper
```

```python
DATADIR = "C:/Users/mobie/OneDrive - Office 365 Fontys/General - AIS zooi/Ass2B/plaatjes

test_data = []

def create_test_data():
    for category in catagories:

        path = os.path.join(DATADIR,category)
        class_num = catagories.index(category)

        for img in tqdm(os.listdir(path)):
            try:
                img_array = cv2.imread(os.path.join(path,img))
                img_array = cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB)


                new_array = cv2.resize(img_array, (32, 32))
                test_data.append([new_array, class_num])

            except Exception as e:
                pass


    plt.imshow(img_array)
    plt.show()

    plt.imshow(new_array)
    plt.show()

create_test_data()
```

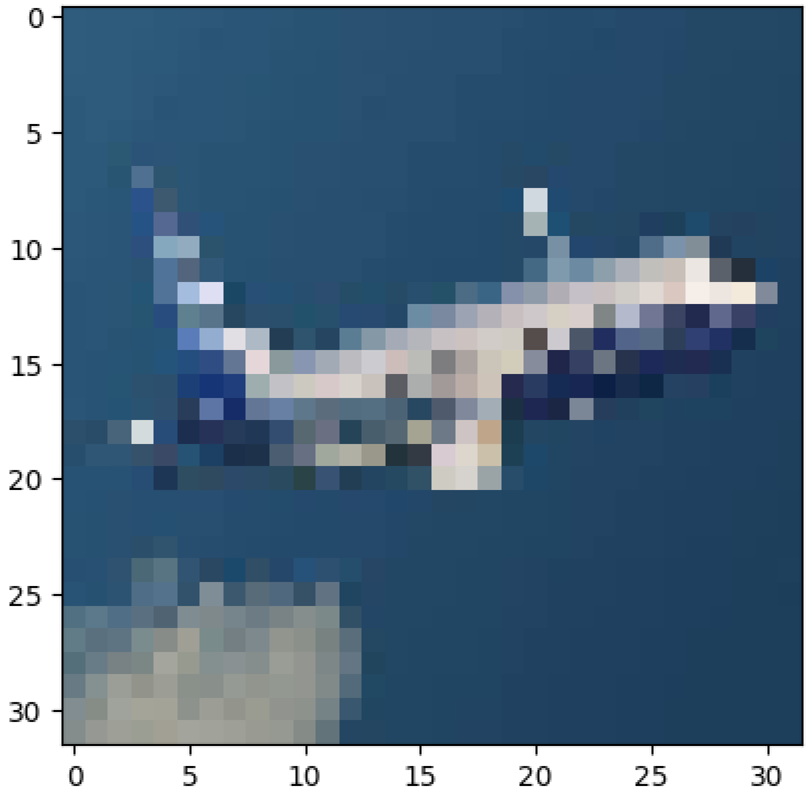
```
100%|██████████| 10/10 [00:00<00:00, 75.07it/s]
100%|██████████| 10/10 [00:00<00:00, 557.04it/s]
100%|██████████| 10/10 [00:00<00:00, 589.41it/s]
100%|██████████| 10/10 [00:01<00:00,  8.29it/s]
100%|██████████| 10/10 [00:00<00:00, 626.70it/s]
100%|██████████| 10/10 [00:00<00:00, 626.63it/s]
100%|██████████| 10/10 [00:00<00:00, 455.75it/s]
100%|██████████| 10/10 [00:00<00:00, 668.45it/s]
100%|██████████| 10/10 [00:00<00:00, 668.45it/s]
100%|██████████| 10/10 [00:00<00:00, 626.71it/s]
```


Here we import our own data set. An for loop iterates trough all images and resizes them and adds them to an array. The before and after pictures of the resize are shown.

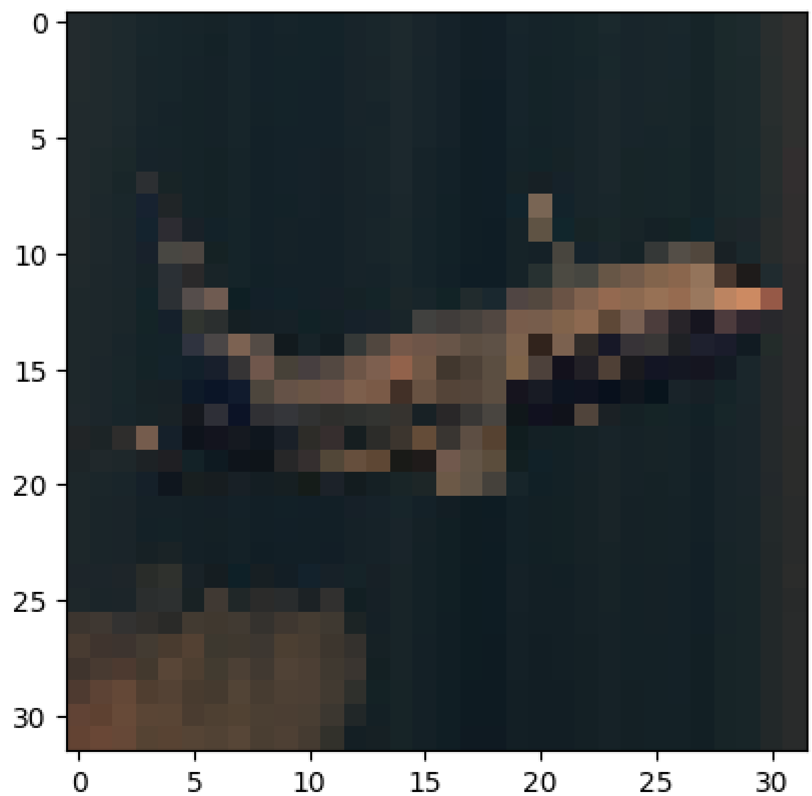
In [15]:
```
X = []
Y = []
y = []
x_test = []

for features,label in test_data:
    X.append(features)
    Y.append(label)
```

```
plt.imshow(X[0], cmap=plt.cm.binary)
plt.show()
print("x = ", X[0])

X = tf.keras.utils.normalize(X, axis=1)


plt.imshow(X[0], cmap=plt.cm.binary)
plt.show()
print("na norm", X[0])
```



```
x =  [[[ 47  92 125]
  [ 46  91 124]
  [ 46  91 124]
  ...
  [ 35  72 101]
  [ 35  72 101]
  [ 34  71 100]]

 [[ 47  92 125]
  [ 46  91 124]
  [ 46  91 124]
  ...
  [ 35  72 101]
  [ 35  72 101]
  [ 34  71 100]]

 [[ 47  92 125]
  [ 46  91 124]
  [ 46  91 124]
  ...
  [ 34  71 100]
  [ 34  71 100]
  [ 34  71 100]]

 ...

 [[104 122 131]
  [133 145 144]
```

```
  [155 158 151]
  ...
  [ 29  63  91]
  [ 29  63  91]
  [ 29  63  91]]

 [[130 141 145]
  [134 140 134]
  [159 163 155]
  ...
  [ 29  63  91]
  [ 29  63  91]
  [ 29  63  91]]

 [[132 141 140]
  [150 152 147]
  [158 159 151]
  ...
  [ 29  63  91]
  [ 29  63  91]
  [ 29  63  91]]]
```




```
na norm [[[0.13840901 0.17216424 0.18154668]
  [0.12656673 0.16713743 0.17910325]
  [0.11706388 0.16291718 0.17844239]
  ...
  [0.11534082 0.16677036 0.17759987]
  [0.16175473 0.18305742 0.18260134]
  [0.19150144 0.19144074 0.18632282]]

 [[0.13840901 0.17216424 0.18154668]
  [0.12656673 0.16713743 0.17910325]
  [0.11706388 0.16291718 0.17844239]
  ...
  [0.11534082 0.16677036 0.17759987]
  [0.16175473 0.18305742 0.18260134]
  [0.19150144 0.19144074 0.18632282]]

 [[0.13840901 0.17216424 0.18154668]
  [0.12656673 0.16713743 0.17910325]
```

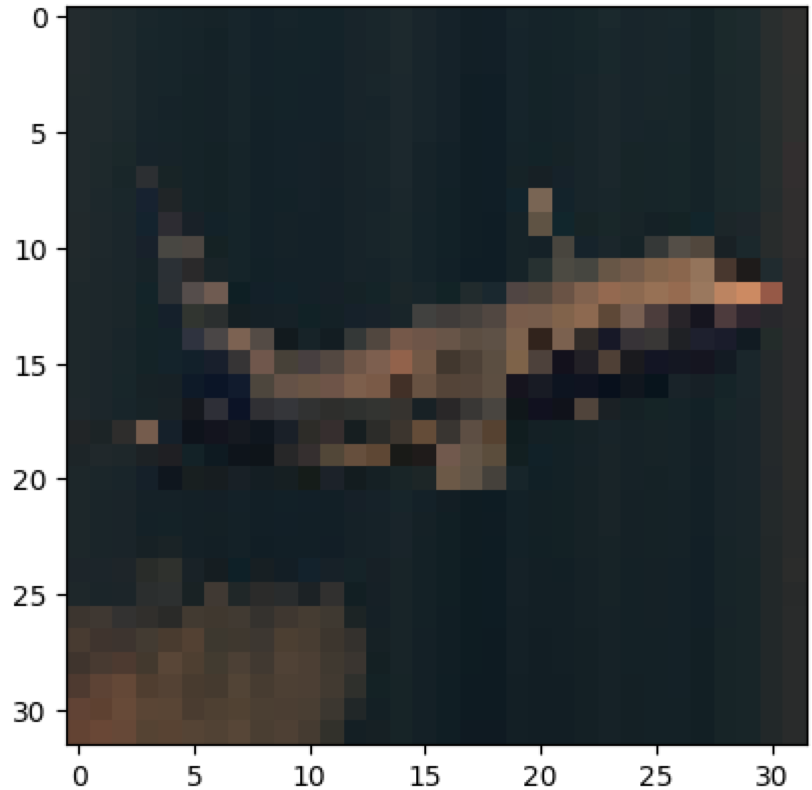
```
  [0.11706388 0.16291718 0.17844239]
  ...
  [0.11204537 0.1644541  0.17584145]
  [0.15713316 0.18051496 0.1807934 ]
  [0.19150144 0.19144074 0.18632282]]

 ...

 [[0.30626675 0.22830475 0.19026092]
  [0.36594295 0.26631788 0.20799087]
  [0.39445439 0.28286719 0.21729678]
  ...
  [0.09556811 0.14592406 0.16001572]
  [0.13402534 0.16017524 0.164522  ]
  [0.16333946 0.16986995 0.16955377]]

 [[0.38283344 0.26386041 0.21059415]
  [0.3686944  0.25713451 0.19354706]
  [0.40463386 0.29181868 0.22305299]
  ...
  [0.09556811 0.14592406 0.16001572]
  [0.13402534 0.16017524 0.164522  ]
  [0.16333946 0.16986995 0.16955377]]

 [[0.38872319 0.26386041 0.20333228]
  [0.41271761 0.27917461 0.21232402]
  [0.402089   0.28465749 0.21729678]
  ...
  [0.09556811 0.14592406 0.16001572]
  [0.13402534 0.16017524 0.164522  ]
  [0.16333946 0.16986995 0.16955377]]]
```

Here we split the data into two lists; the pictures and the catagories. After that the images are normalized this means all the values are made between zero and one and the before and after is shown.

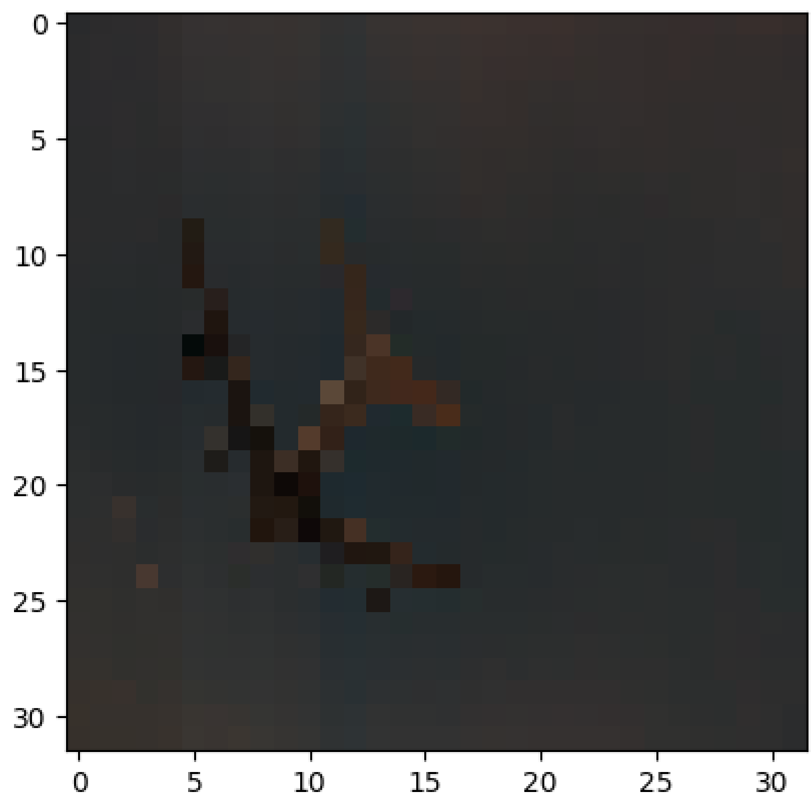
In [14]:

```python
g=0
f=0

for x, y in zip(X, Y):
    print(x.shape)
    plt.imshow(x, cmap=plt.cm.binary)
    plt.show()
    x = np.expand_dims(x, axis=0)
    predictions = new_model.predict([x])
    #print(predictions)
    print("picture shows:    ", catagories[y])
    print("model prediction: ", catagories[np.argmax(predictions)])
    if (y==np.argmax(predictions)):
        g = g+1
        print("correct")
    else:
        f = f+1
        print("wrong")

acc = g/(g+f)
tot = g + f
print("total =", tot)
print("acc =", acc*100,"%")
```

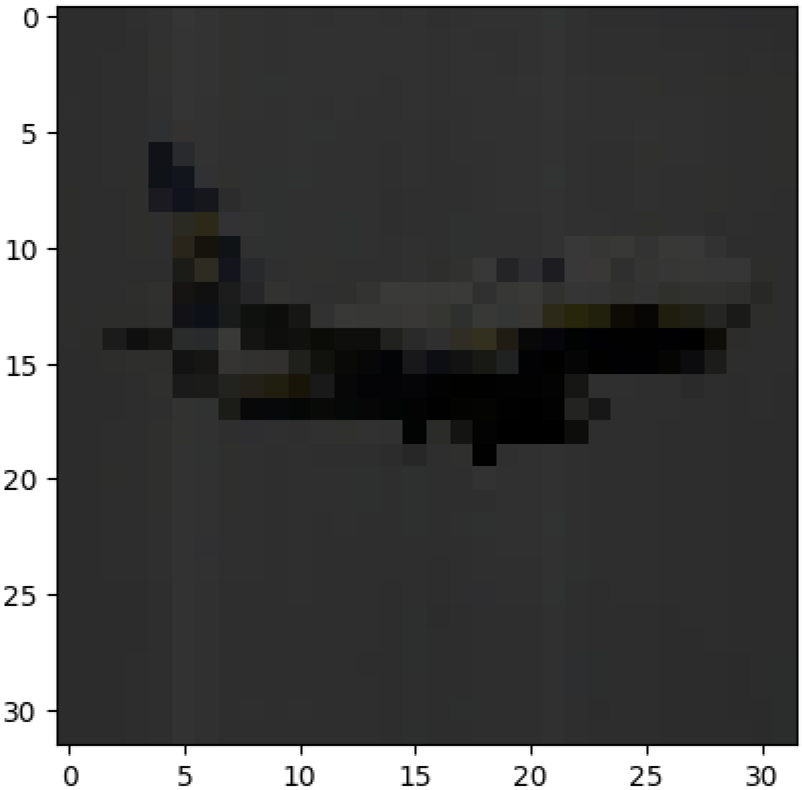
```
(32, 32, 3)
```



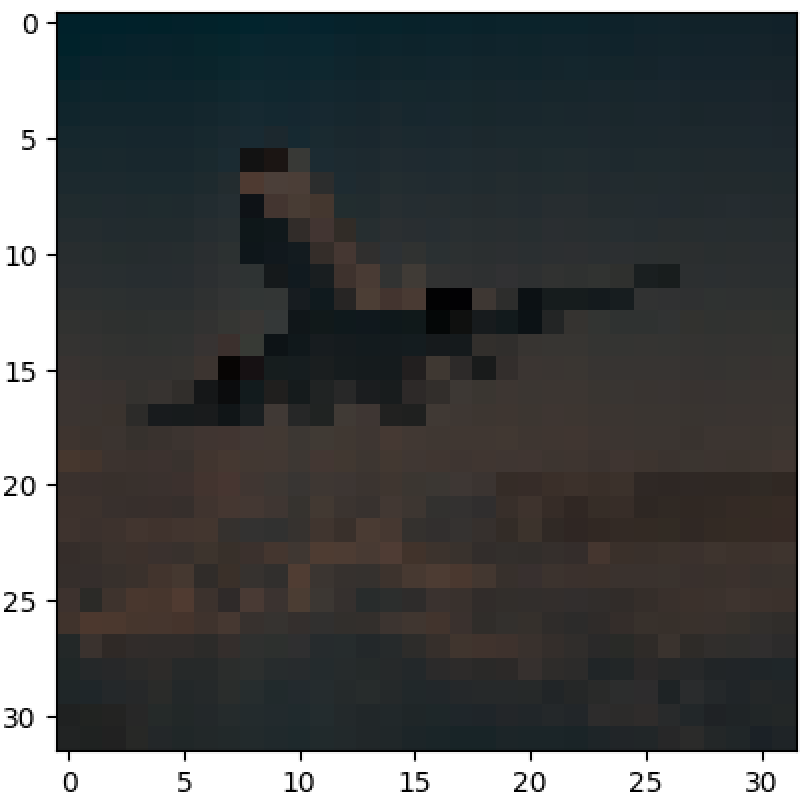

```
1/1 [==============================] - 0s 236ms/step
picture shows:      airplane
model prediction:   airplane
correct
(32, 32, 3)
```



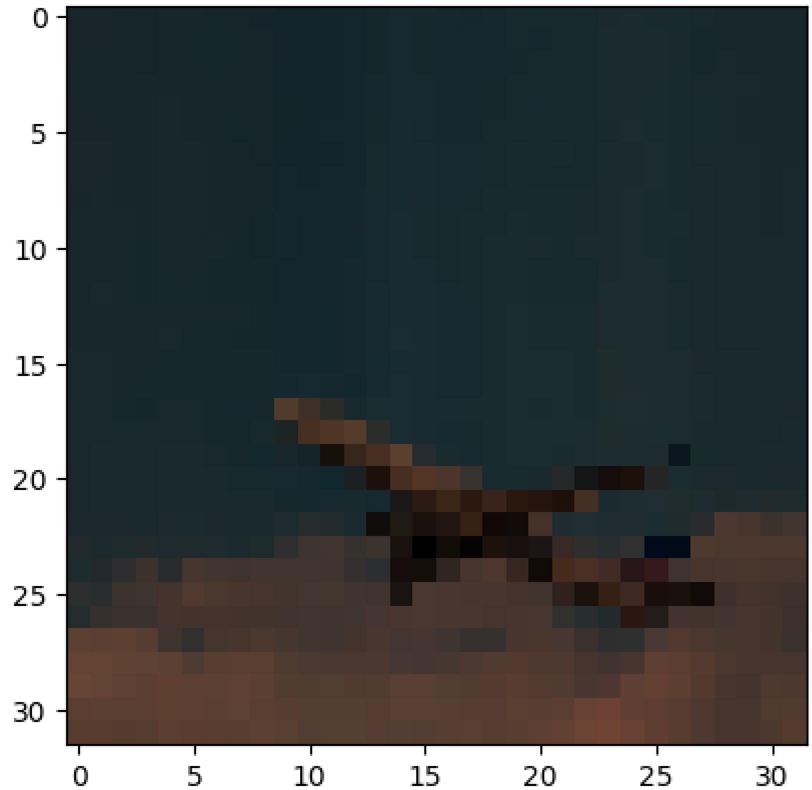

```
1/1 [==============================] - 0s 24ms/step
picture shows:      airplane
model prediction:   airplane
correct
(32, 32, 3)
```



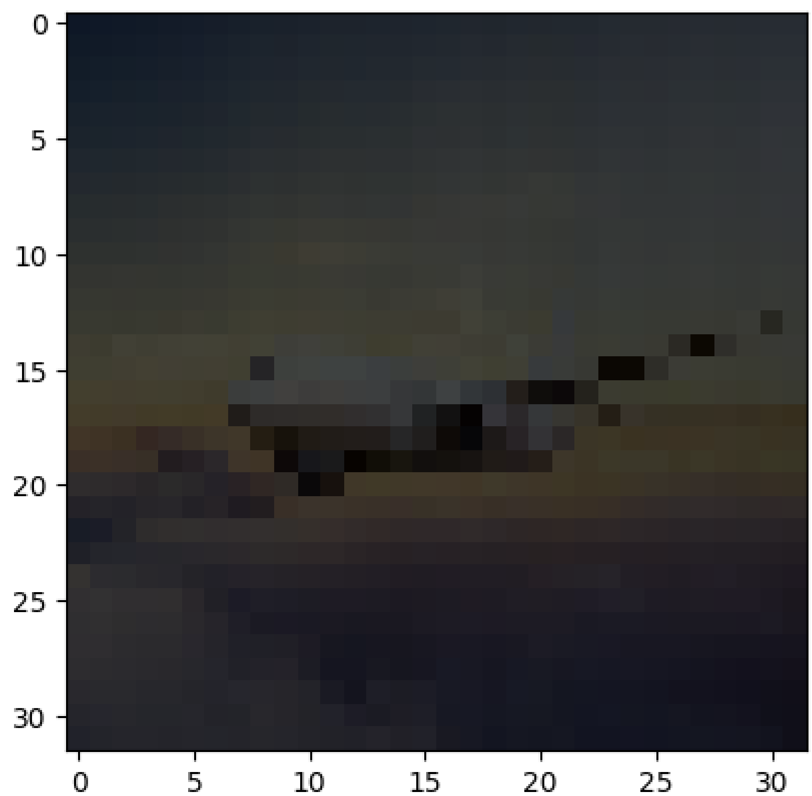
```
1/1 [==============================] - 0s 29ms/step
picture shows:      airplane
model prediction:  airplane
correct
(32, 32, 3)
```



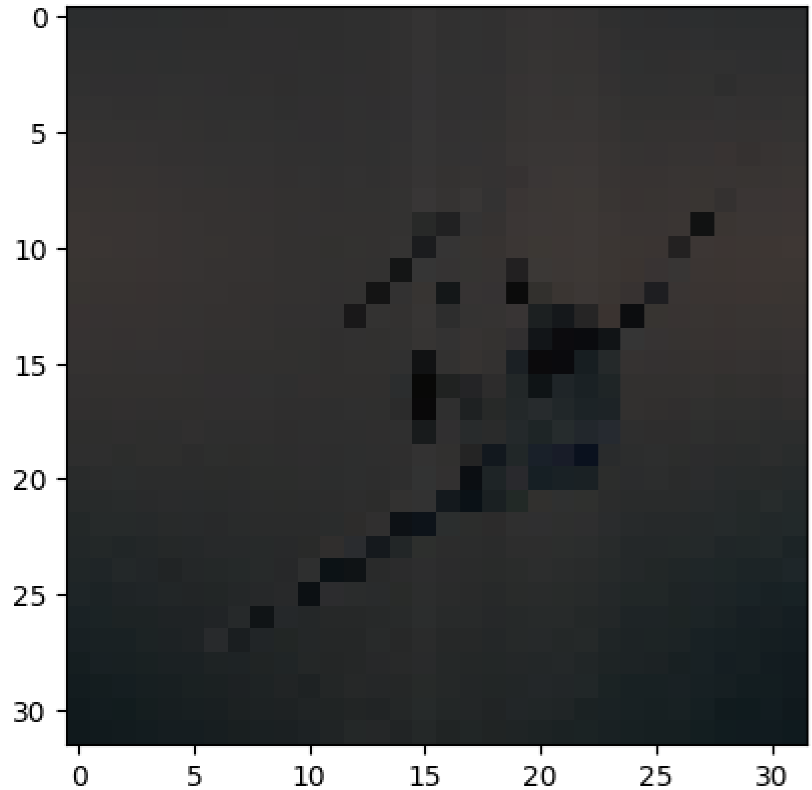
```
1/1 [==============================] - 0s 25ms/step
picture shows:      airplane
model prediction:  airplane
correct
(32, 32, 3)
```



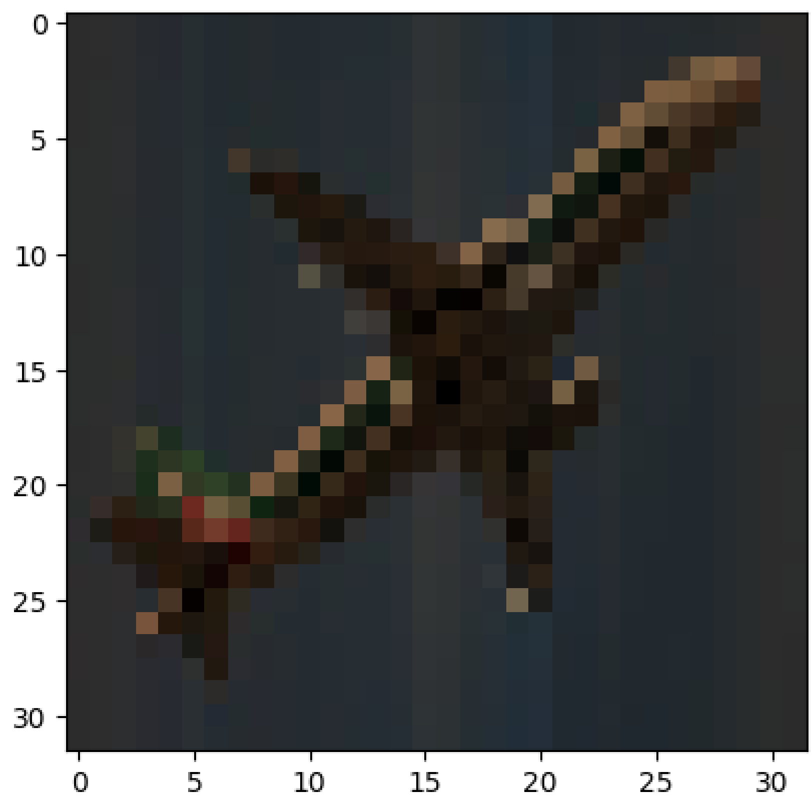
```
1/1 [==============================] - 0s 27ms/step
picture shows:      airplane
model prediction:   airplane
correct
(32, 32, 3)
```



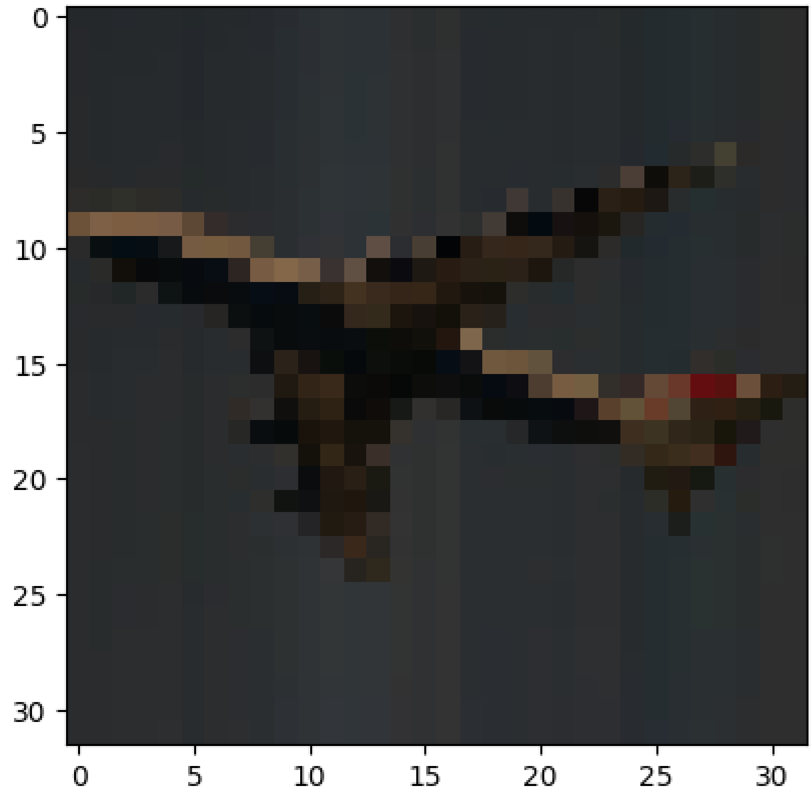
```
1/1 [==============================] - 0s 23ms/step
picture shows:      airplane
model prediction:   airplane
correct
(32, 32, 3)
```



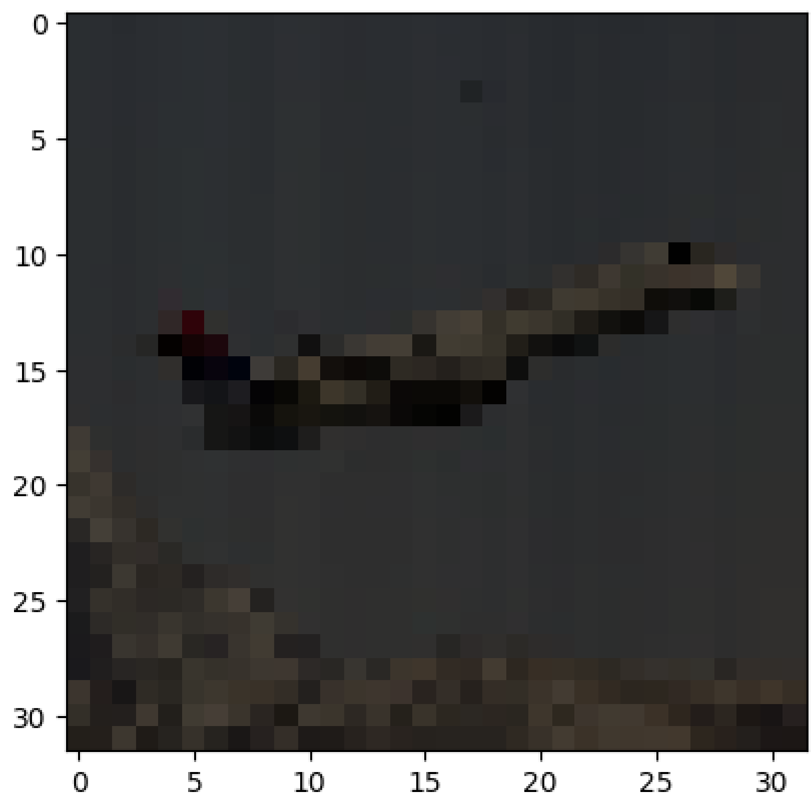

```
1/1 [==============================] - 0s 25ms/step
picture shows:      airplane
model prediction:  airplane
correct
(32, 32, 3)
```



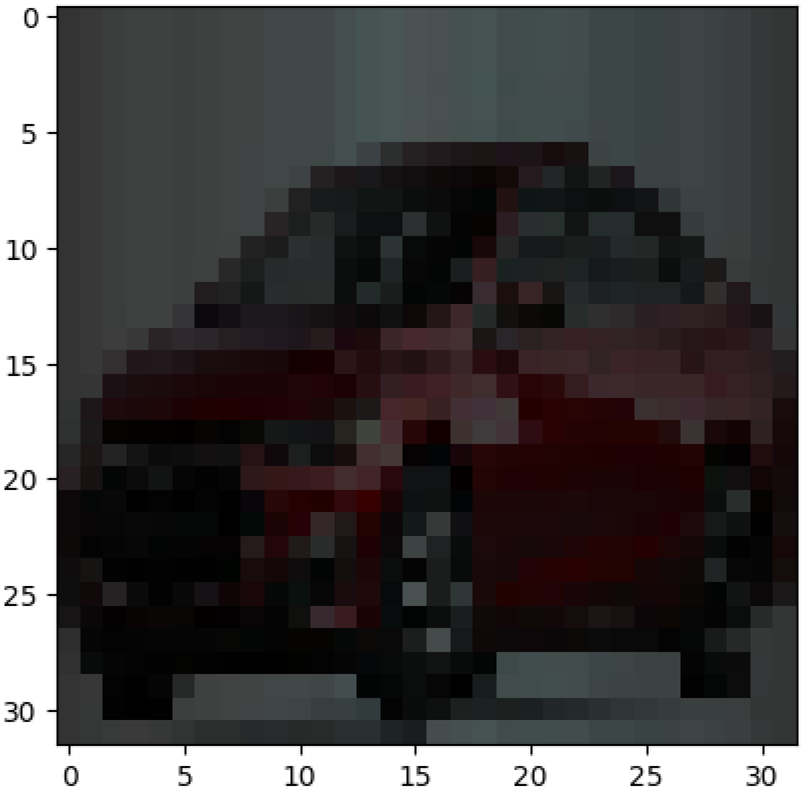

```
1/1 [==============================] - 0s 24ms/step
picture shows:      airplane
model prediction:  airplane
correct
(32, 32, 3)
```



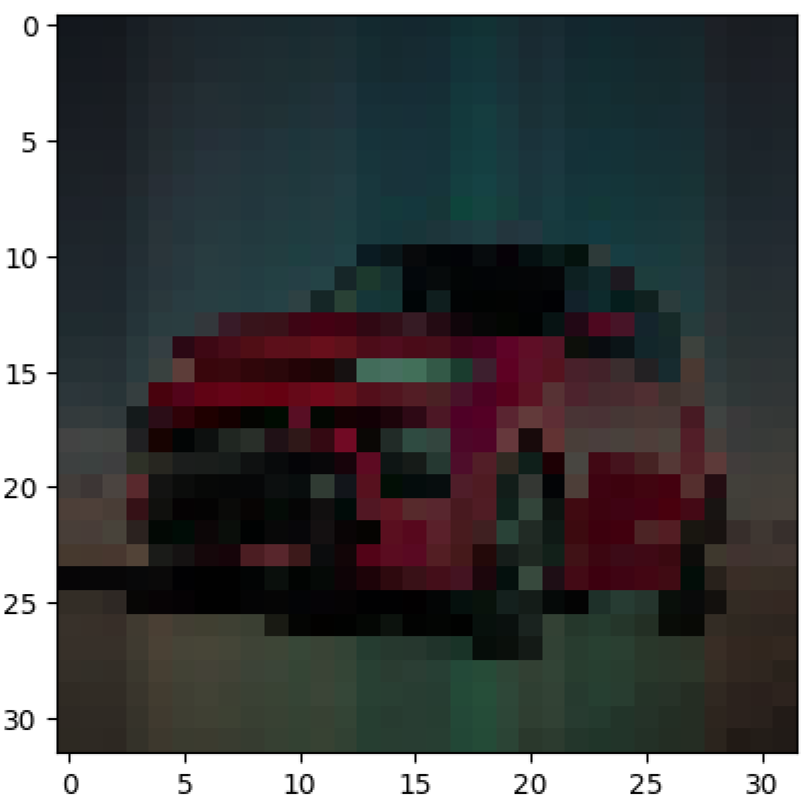

```
1/1 [==============================] - 0s 33ms/step
picture shows:     airplane
model prediction:  airplane
correct
(32, 32, 3)
```



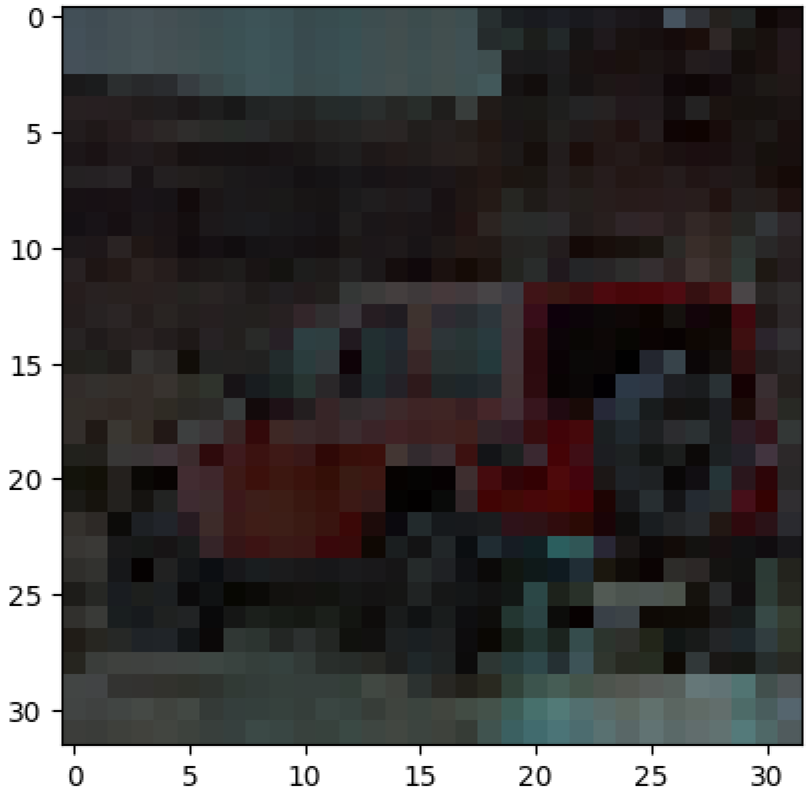

```
1/1 [==============================] - 0s 22ms/step
picture shows:     airplane
model prediction:  airplane
correct
(32, 32, 3)
```



```
1/1 [==============================] - 0s 22ms/step
picture shows:     automobile
model prediction:  automobile
correct
(32, 32, 3)
```

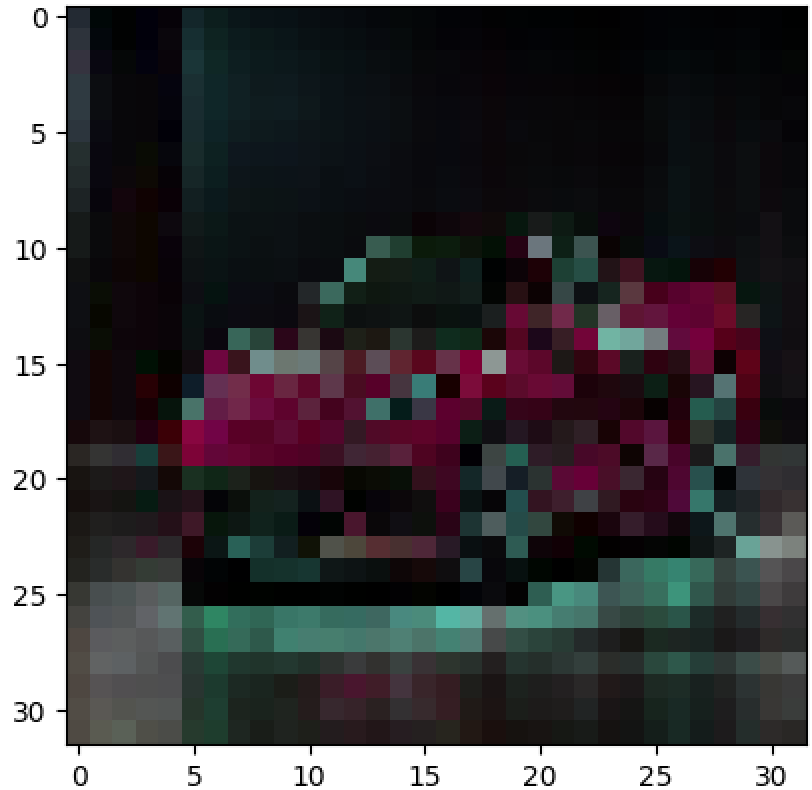


```
1/1 [==============================] - 0s 24ms/step
picture shows:     automobile
model prediction:  automobile
correct
(32, 32, 3)
```

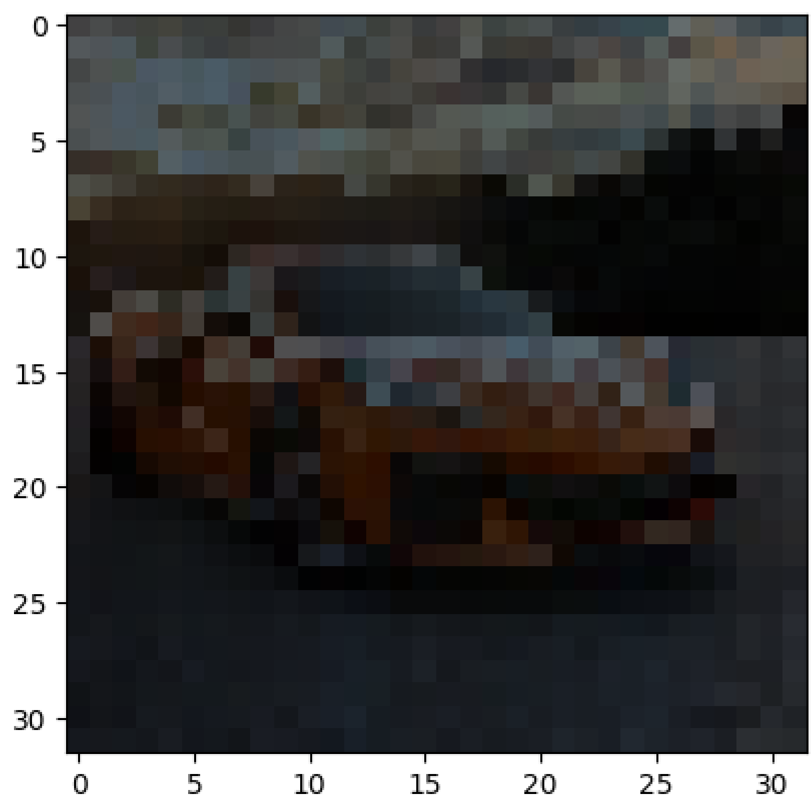


```
1/1 [==============================] - 0s 23ms/step
picture shows:      automobile
model prediction:   truck
wrong
(32, 32, 3)
```

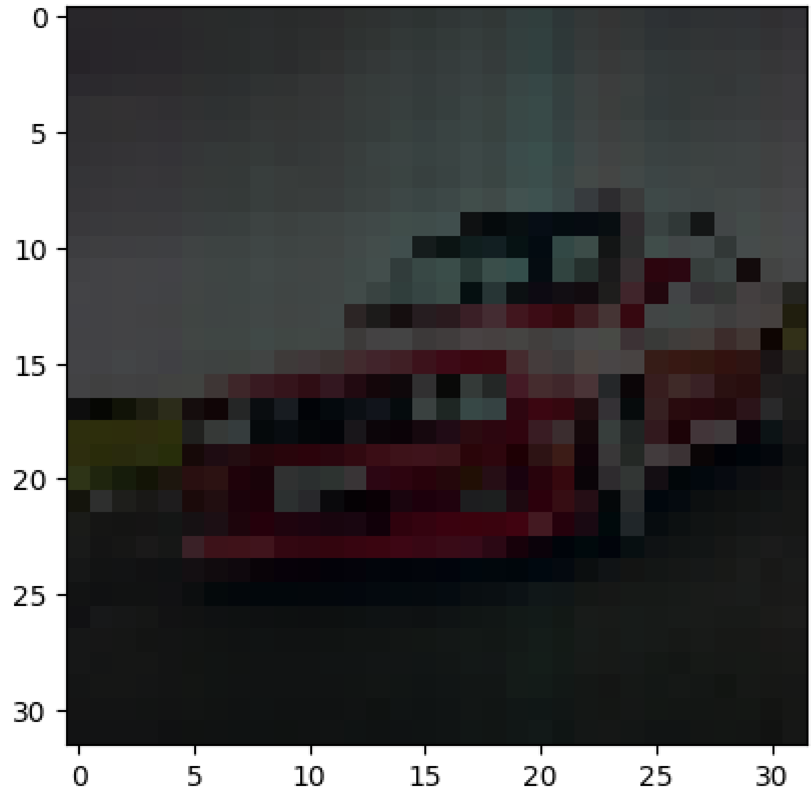
```
1/1 [==============================] - 0s 22ms/step
picture shows:      automobile
model prediction:   automobile
correct
(32, 32, 3)
```



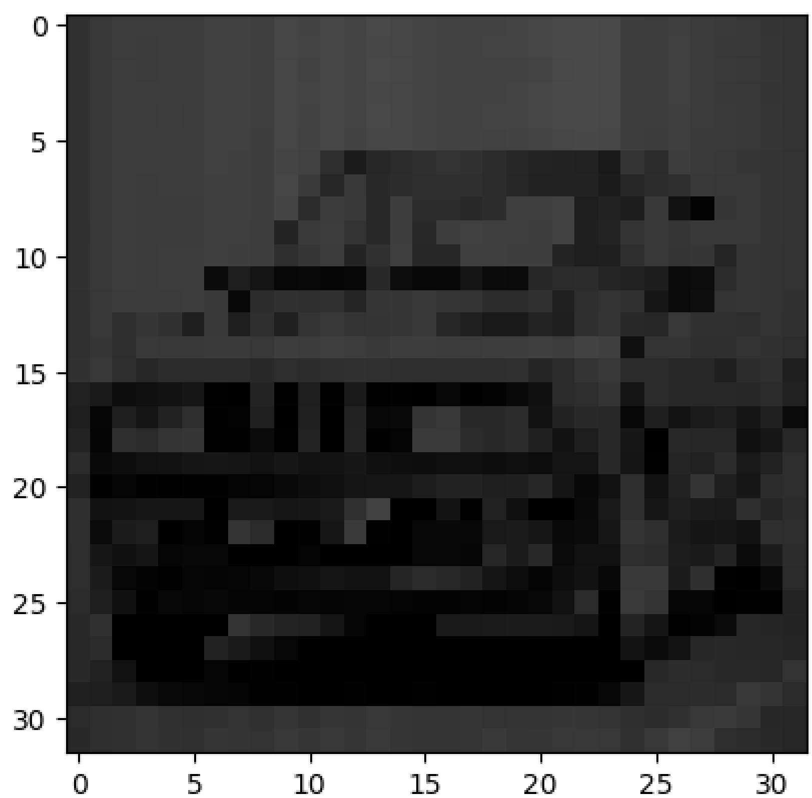
```
1/1 [==============================] - 0s 59ms/step
picture shows:     automobile
model prediction:  automobile
correct
(32, 32, 3)
```



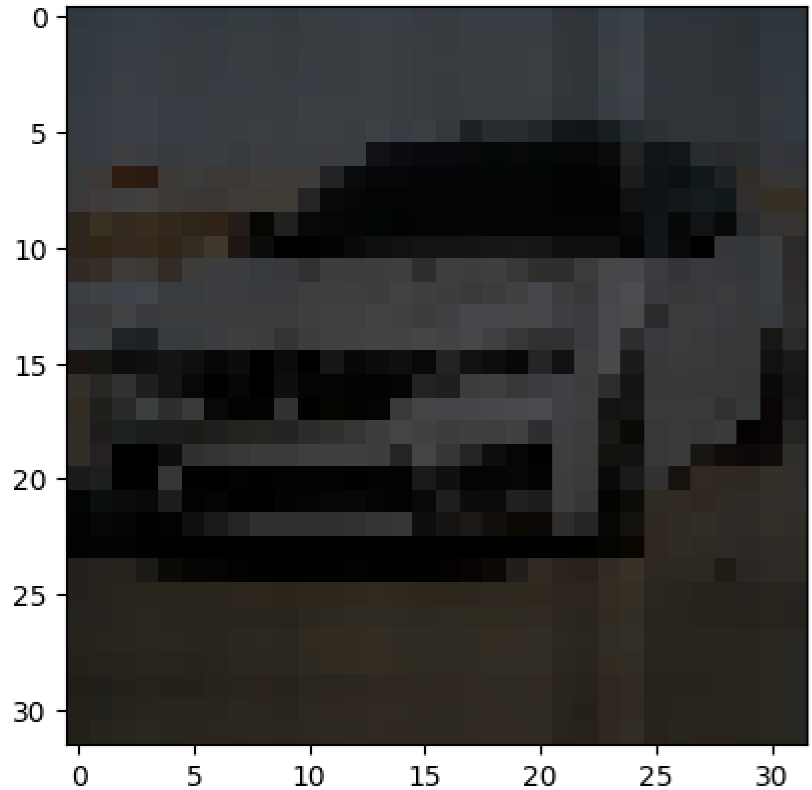
```
1/1 [==============================] - 0s 24ms/step
picture shows:     automobile
model prediction:  truck
wrong
(32, 32, 3)
```



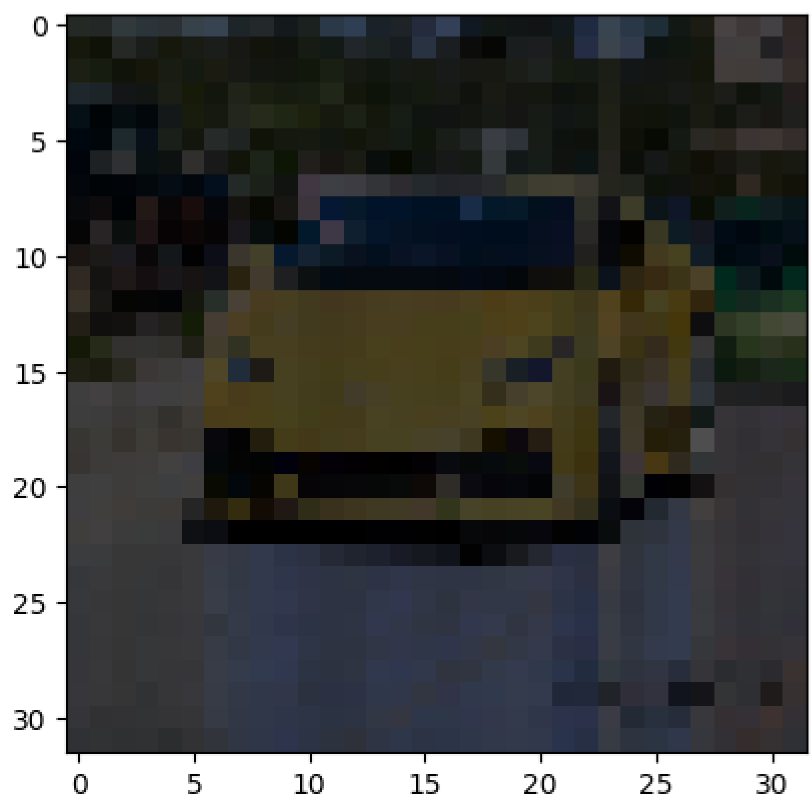

```
1/1 [==============================] - 0s 23ms/step
picture shows:     automobile
model prediction:  automobile
correct
(32, 32, 3)
```



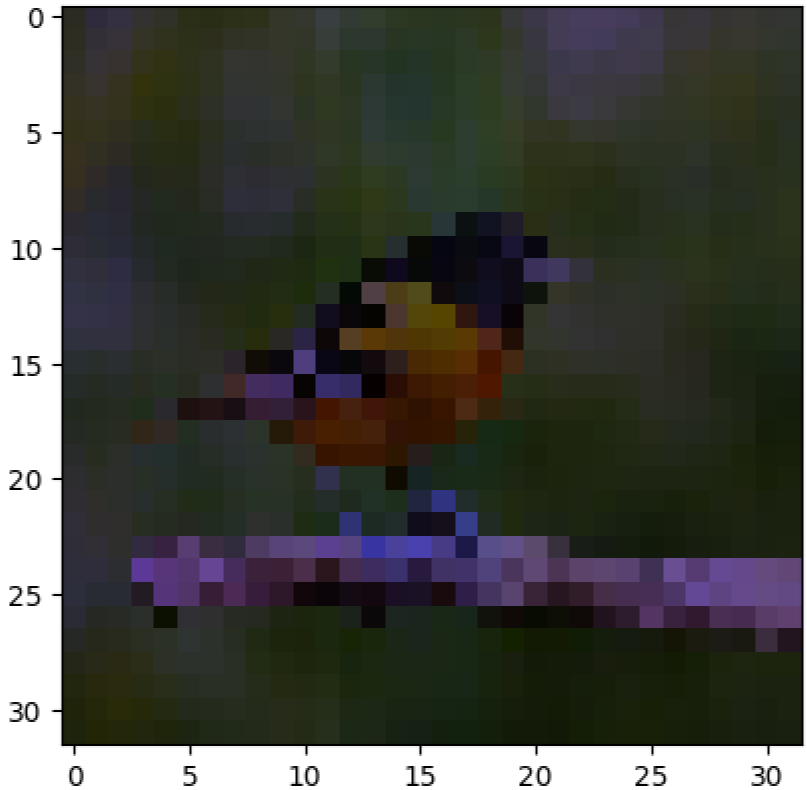

```
1/1 [==============================] - 0s 23ms/step
picture shows:     automobile
model prediction:  truck
wrong
(32, 32, 3)
```



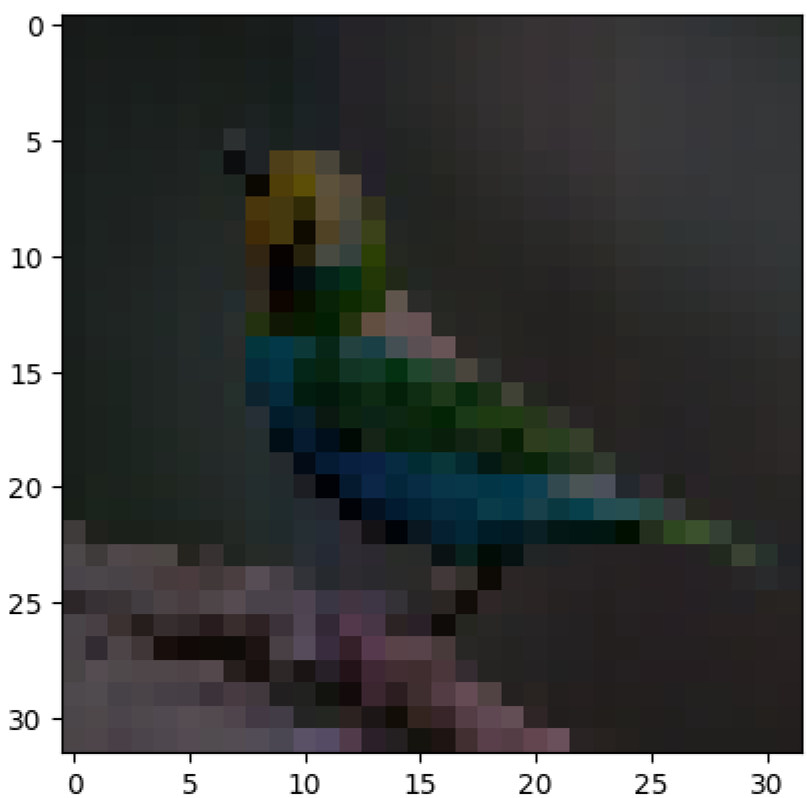
```
1/1 [==============================] - 0s 24ms/step
picture shows:      automobile
model prediction:   automobile
correct
(32, 32, 3)
```



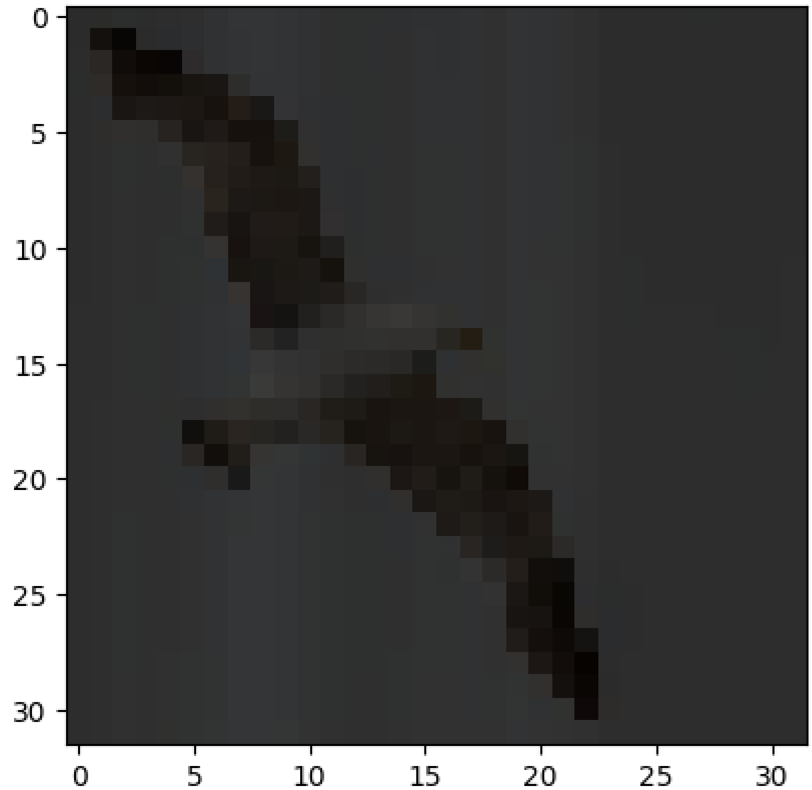
```
1/1 [==============================] - 0s 28ms/step
picture shows:      automobile
model prediction:   automobile
correct
(32, 32, 3)
```



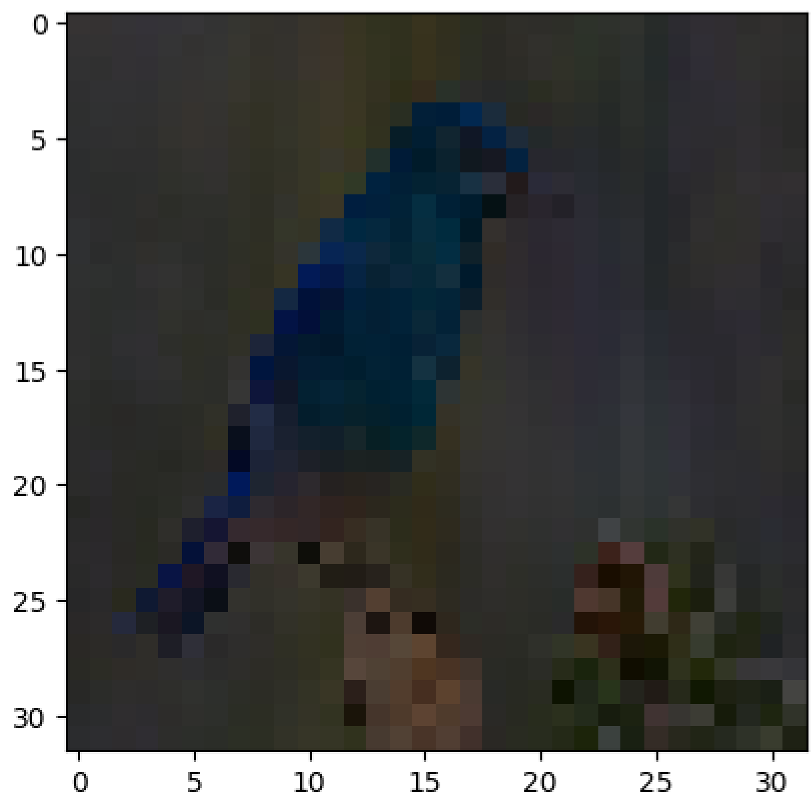
```
1/1 [==============================] - 0s 23ms/step
picture shows:     bird
model prediction:  bird
correct
(32, 32, 3)
```



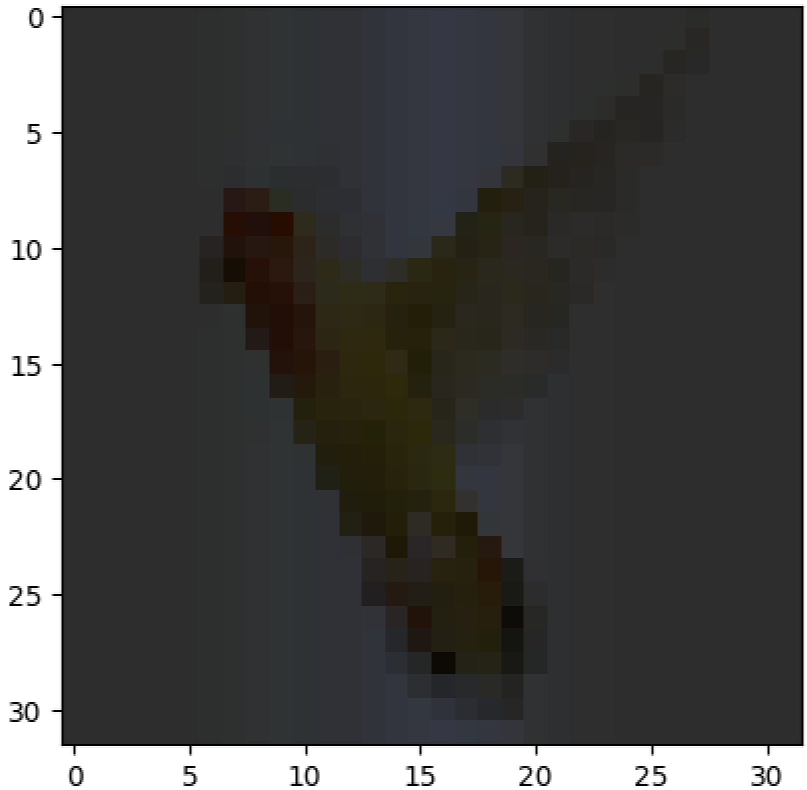
```
1/1 [==============================] - 0s 22ms/step
picture shows:     bird
model prediction:  bird
correct
(32, 32, 3)
```



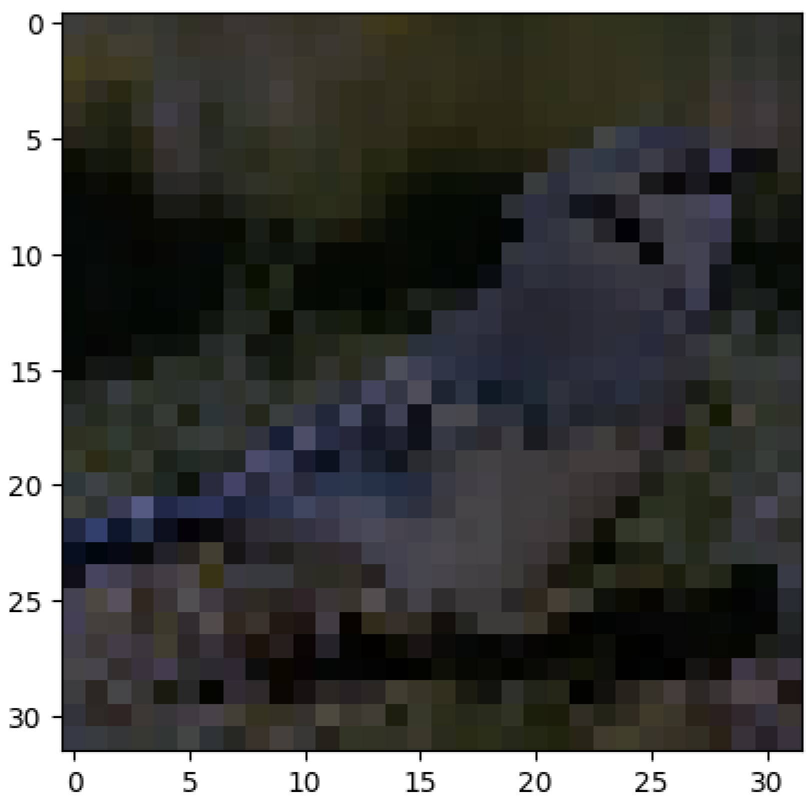

```
1/1 [==============================] - 0s 27ms/step
picture shows:     bird
model prediction:  airplane
wrong
(32, 32, 3)
```



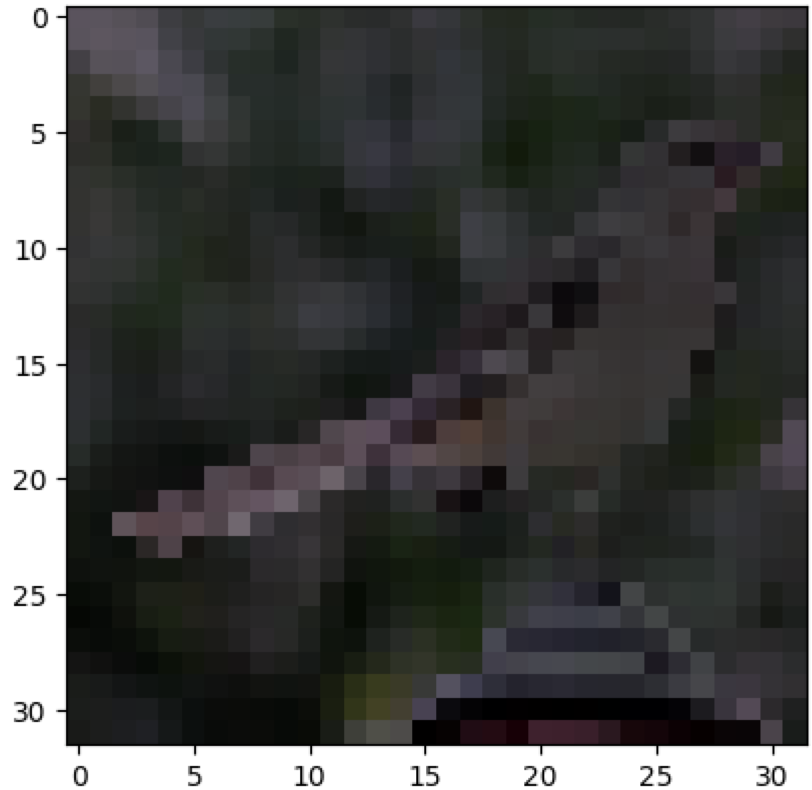

```
1/1 [==============================] - 0s 26ms/step
picture shows:     bird
model prediction:  bird
correct
(32, 32, 3)
```



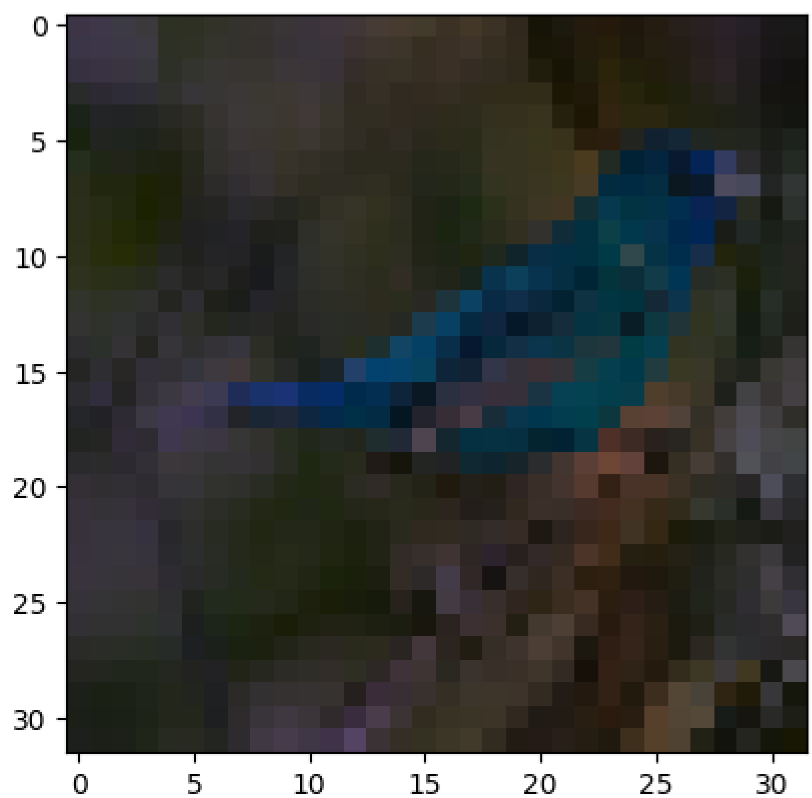
```
1/1 [==============================] - 0s 26ms/step
picture shows:     bird
model prediction:  bird
correct
(32, 32, 3)
```



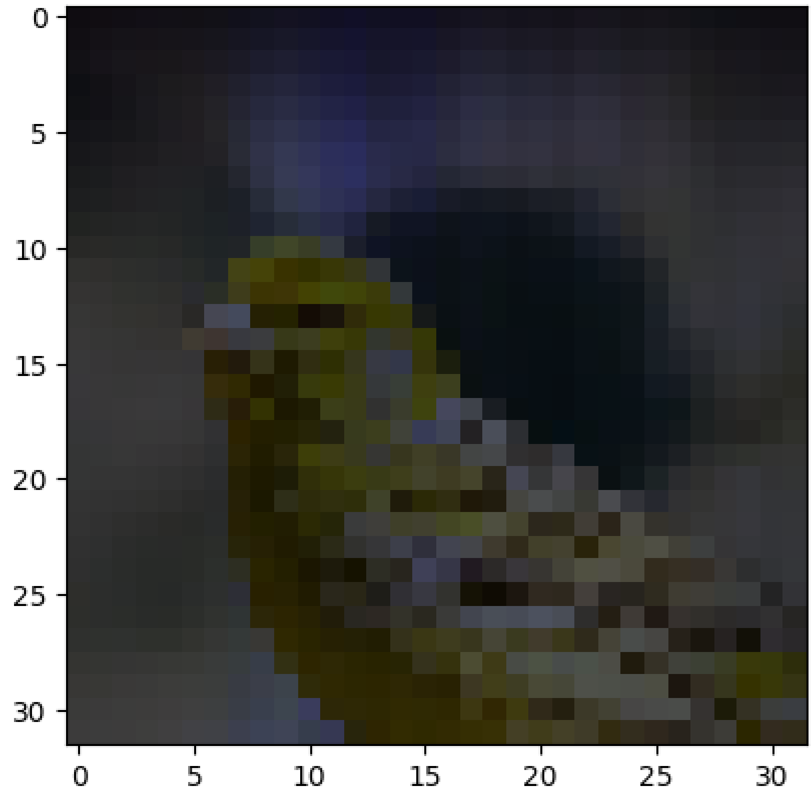
```
1/1 [==============================] - 0s 25ms/step
picture shows:     bird
model prediction:  automobile
wrong
(32, 32, 3)
```



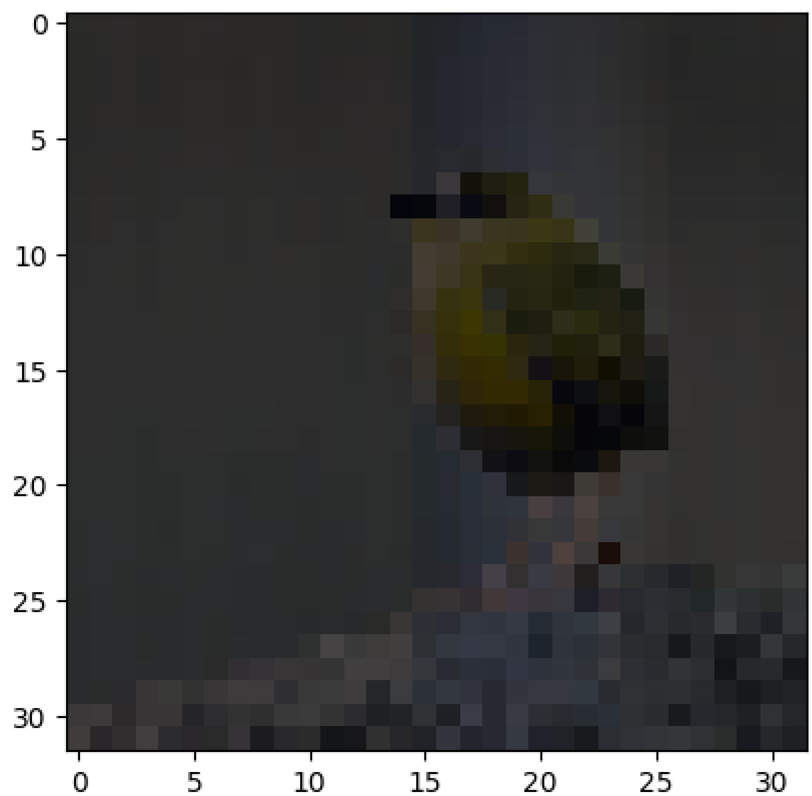

```
1/1 [==============================] - 0s 24ms/step
picture shows:      bird
model prediction:  bird
correct
(32, 32, 3)
```



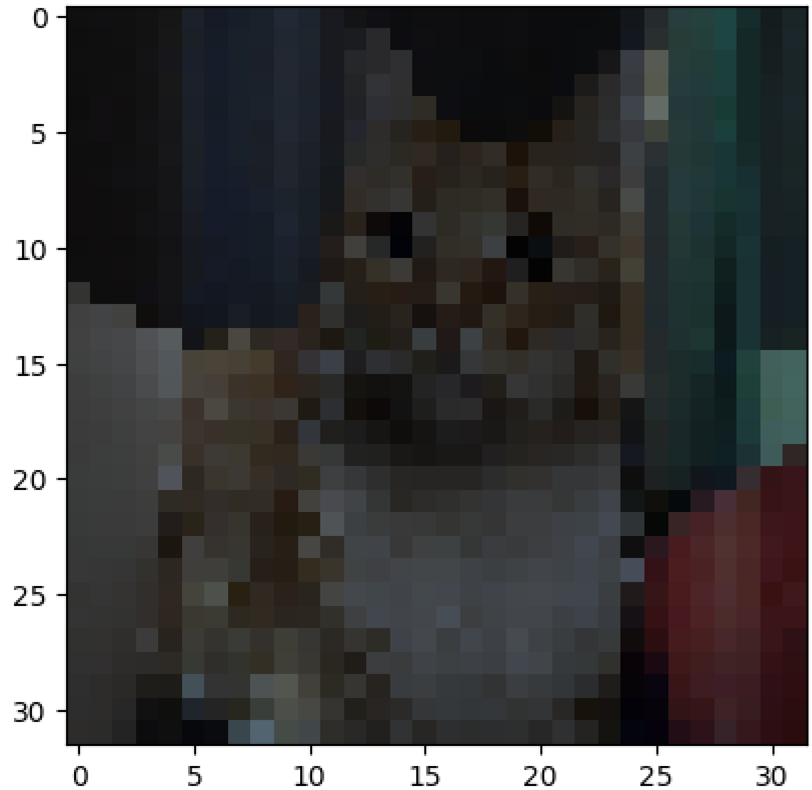

```
1/1 [==============================] - 0s 23ms/step
picture shows:      bird
model prediction:  airplane
wrong
(32, 32, 3)
```



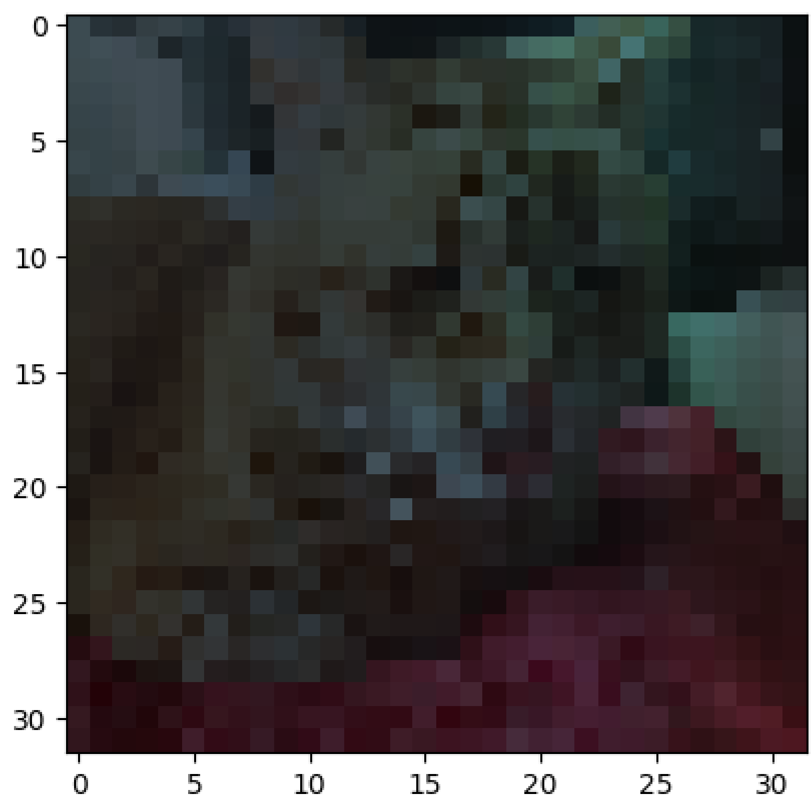

```
1/1 [==============================] - 0s 30ms/step
picture shows:      bird
model prediction:  frog
wrong
(32, 32, 3)
```



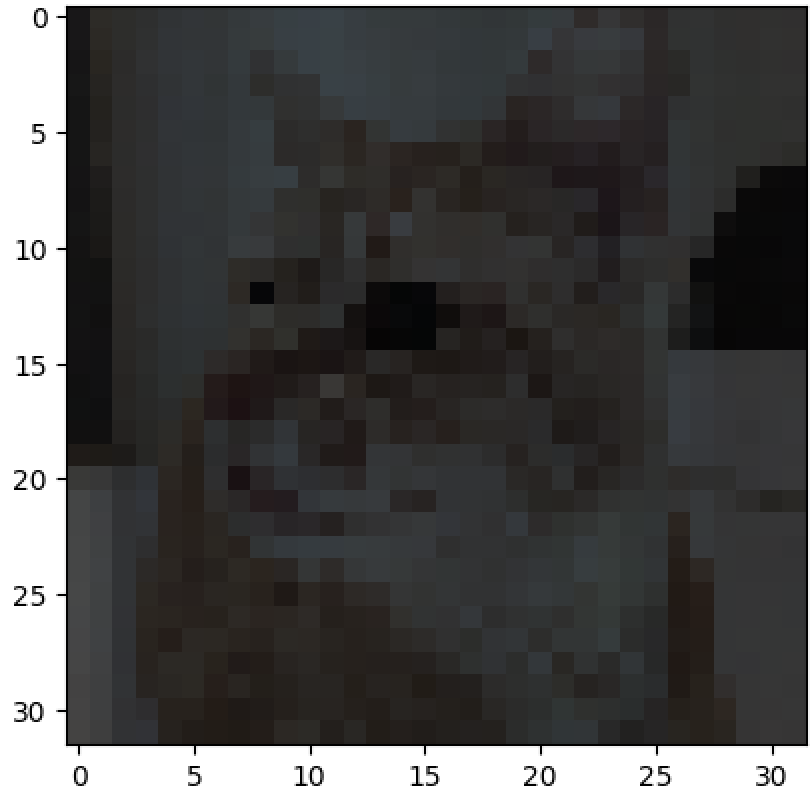

```
1/1 [==============================] - 0s 26ms/step
picture shows:      bird
model prediction:  bird
correct
(32, 32, 3)
```



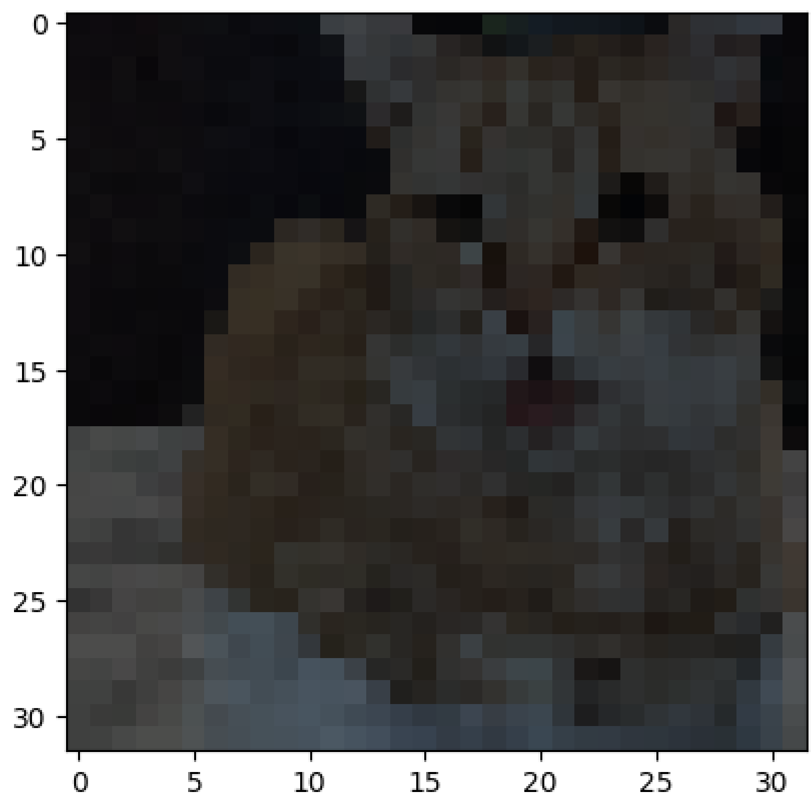

```
1/1 [==============================] - 0s 24ms/step
picture shows:      cat
model prediction:   dog
wrong
(32, 32, 3)
```



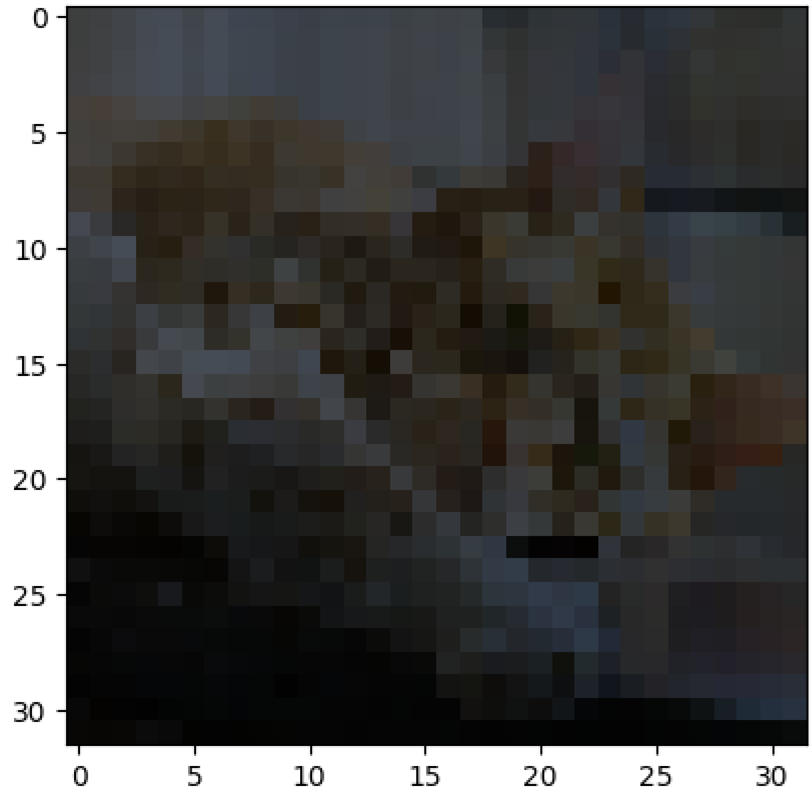

```
1/1 [==============================] - 0s 25ms/step
picture shows:      cat
model prediction:   truck
wrong
(32, 32, 3)
```



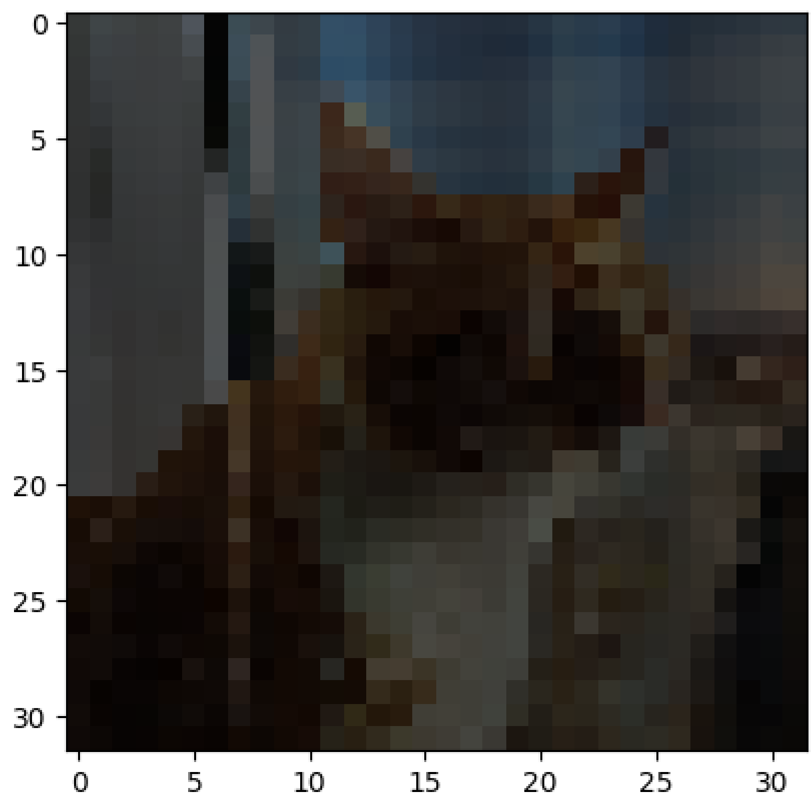
```
1/1 [==============================] - 0s 26ms/step
picture shows:      cat
model prediction:  cat
correct
(32, 32, 3)
```



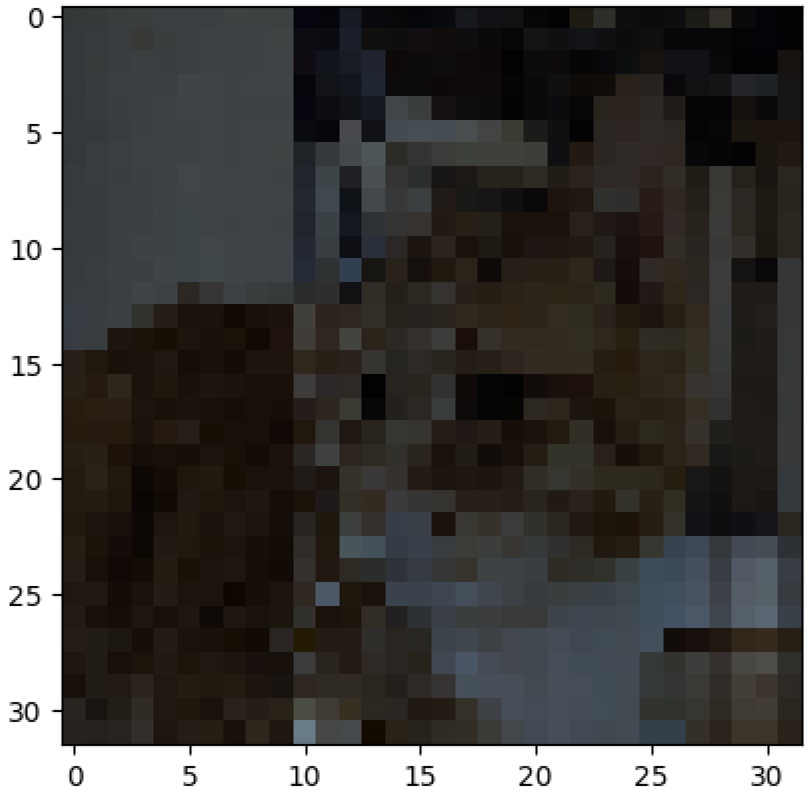
```
1/1 [==============================] - 0s 26ms/step
picture shows:      cat
model prediction:  dog
wrong
(32, 32, 3)
```



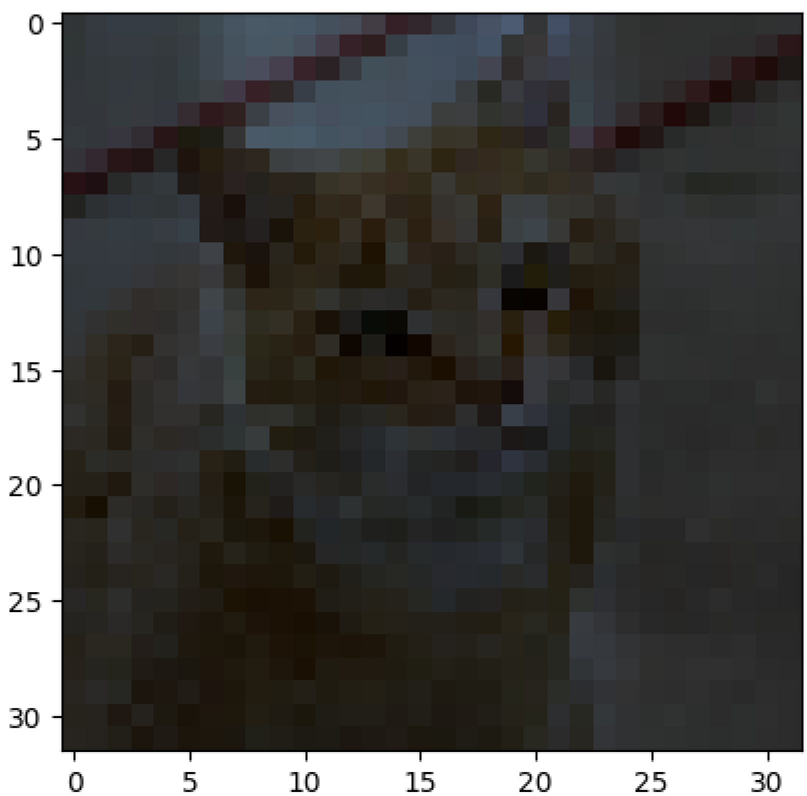
```
1/1 [==============================] - 0s 26ms/step
picture shows:      cat
model prediction:  frog
wrong
(32, 32, 3)
```



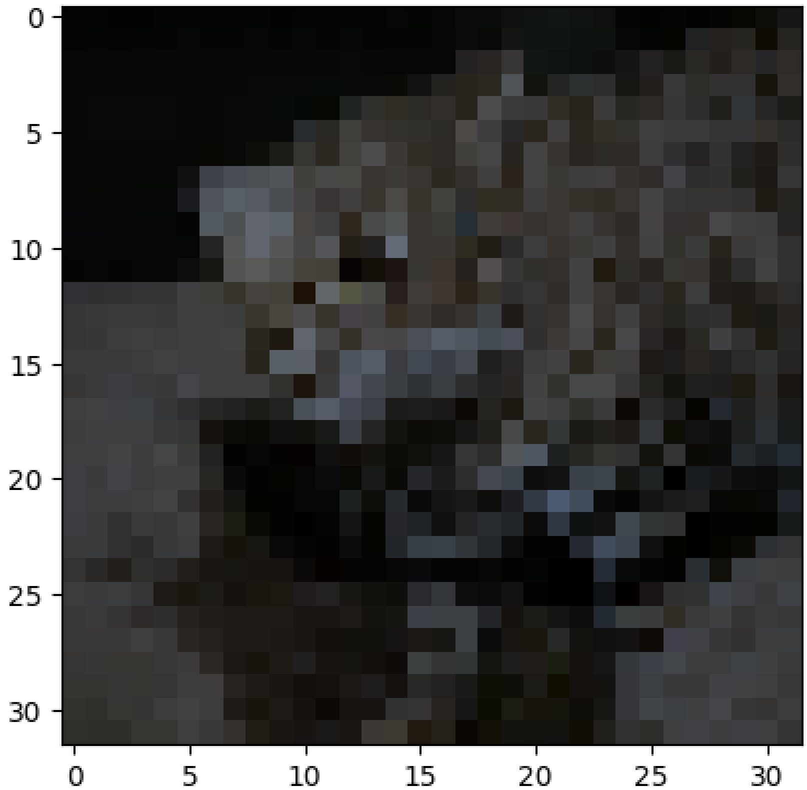
```
1/1 [==============================] - 0s 24ms/step
picture shows:      cat
model prediction:  cat
correct
(32, 32, 3)
```



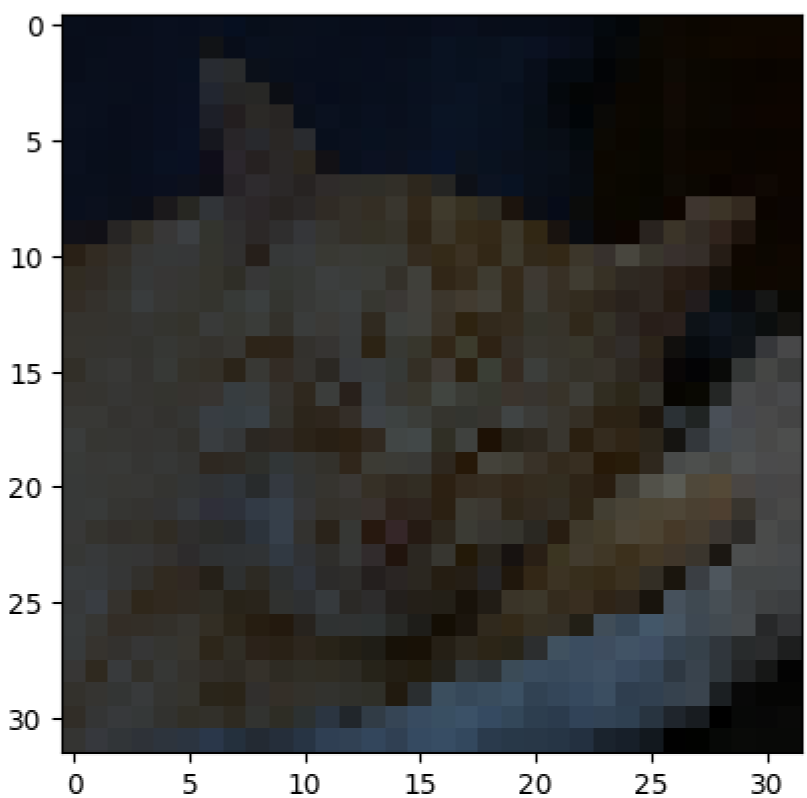

```
1/1 [==============================] - 0s 21ms/step
picture shows:      cat
model prediction:  cat
correct
(32, 32, 3)
```



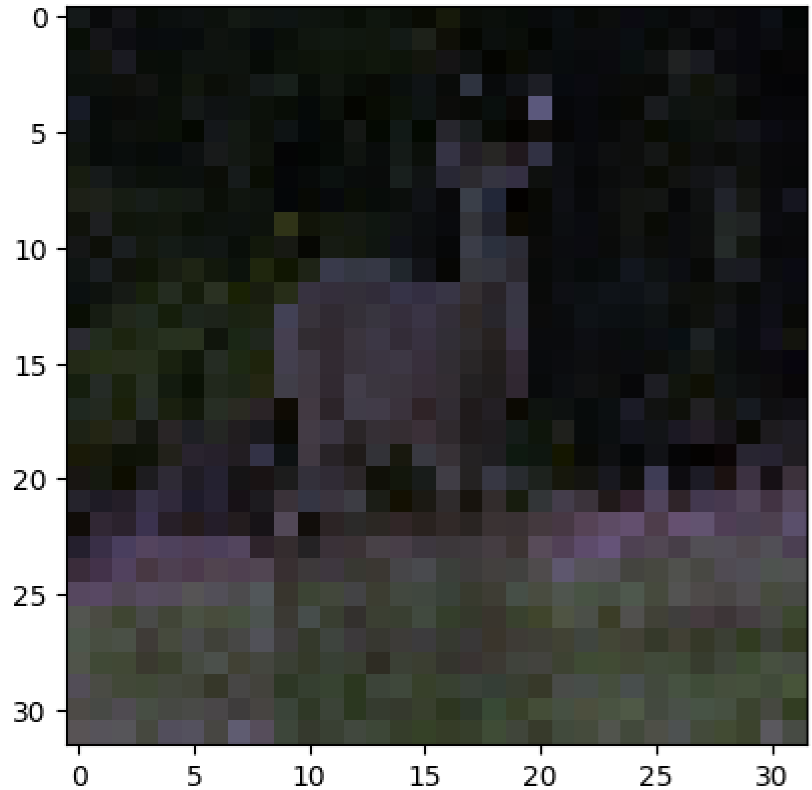

```
1/1 [==============================] - 0s 23ms/step
picture shows:      cat
model prediction:  deer
wrong
(32, 32, 3)
```



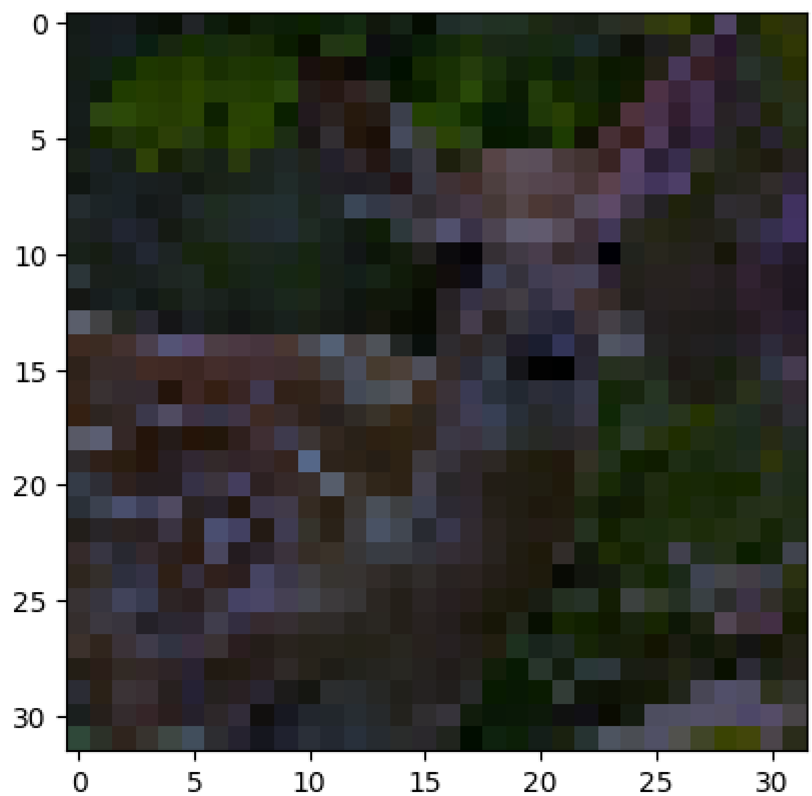

```
1/1 [==============================] - 0s 26ms/step
picture shows:      cat
model prediction:  frog
wrong
(32, 32, 3)
```



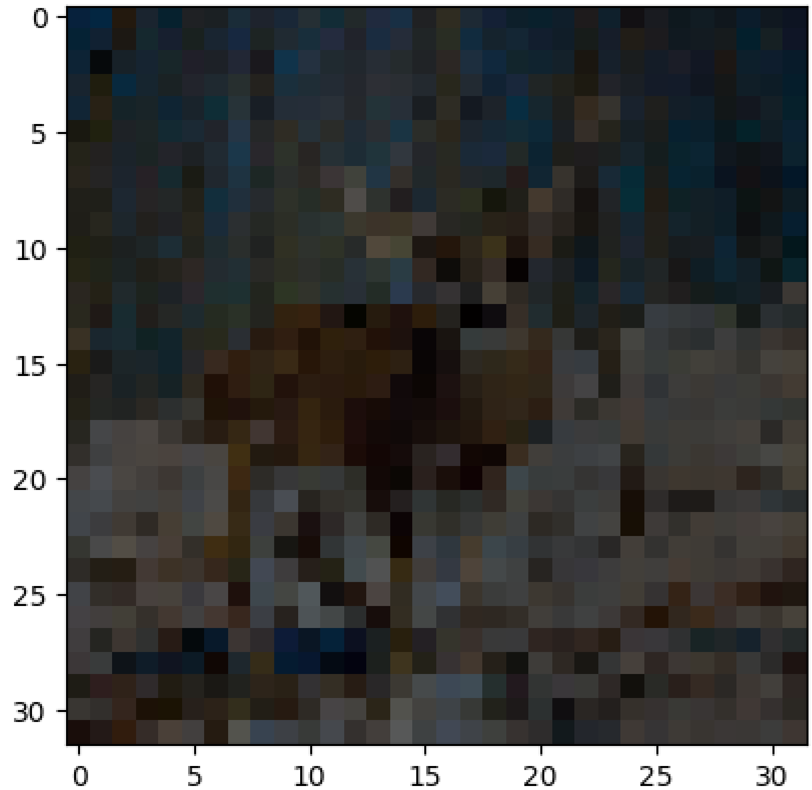

```
1/1 [==============================] - 0s 23ms/step
picture shows:      cat
model prediction:  frog
wrong
(32, 32, 3)
```



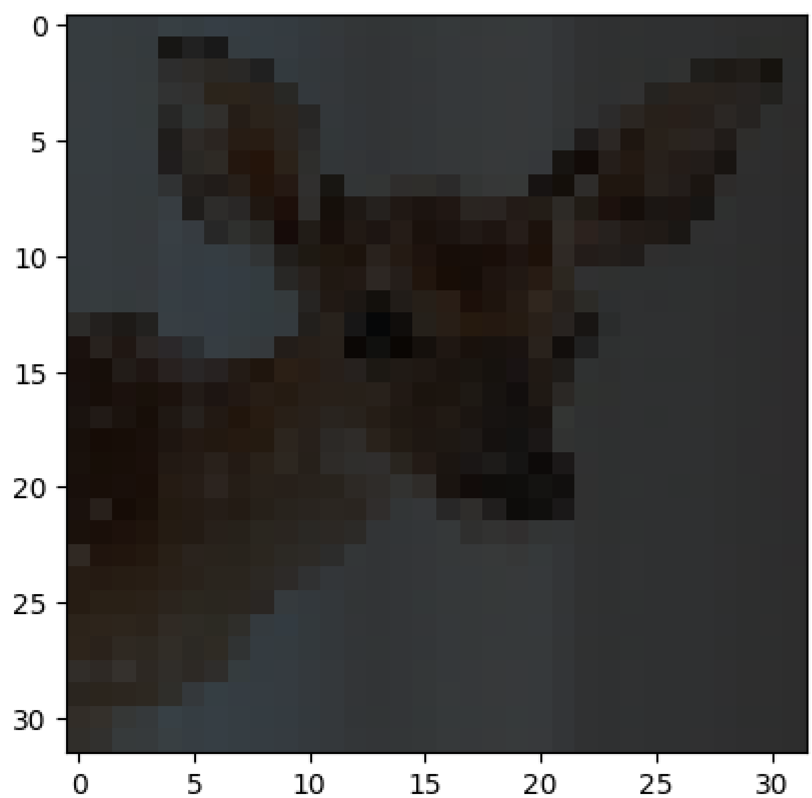
```
1/1 [==============================] - 0s 25ms/step
picture shows:     deer
model prediction:  bird
wrong
(32, 32, 3)
```



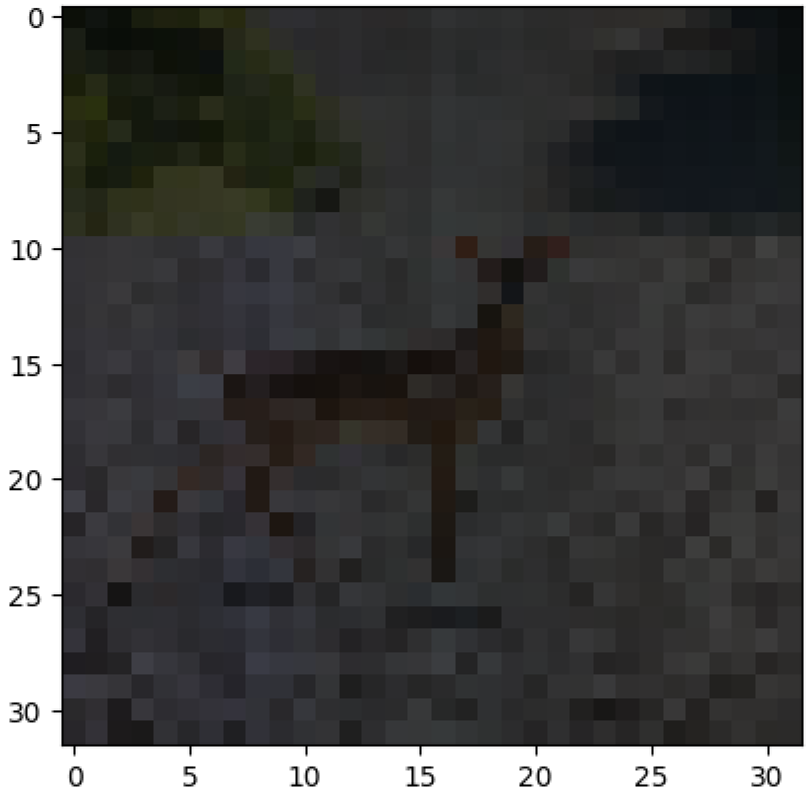
```
1/1 [==============================] - 0s 27ms/step
picture shows:     deer
model prediction:  horse
wrong
(32, 32, 3)
```



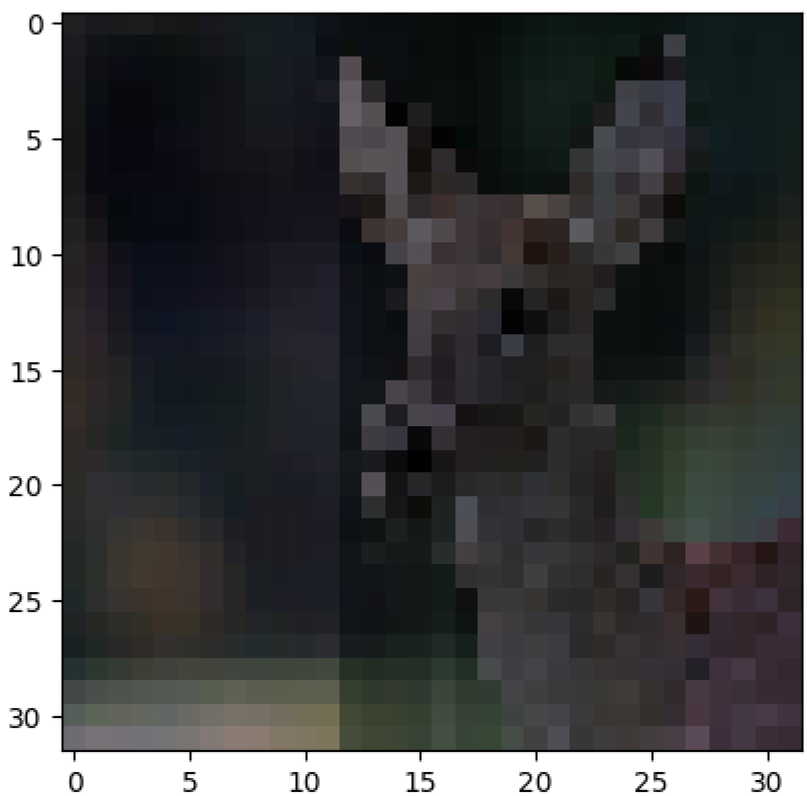

```
1/1 [==============================] - 0s 24ms/step
picture shows:     deer
model prediction:  frog
wrong
(32, 32, 3)
```



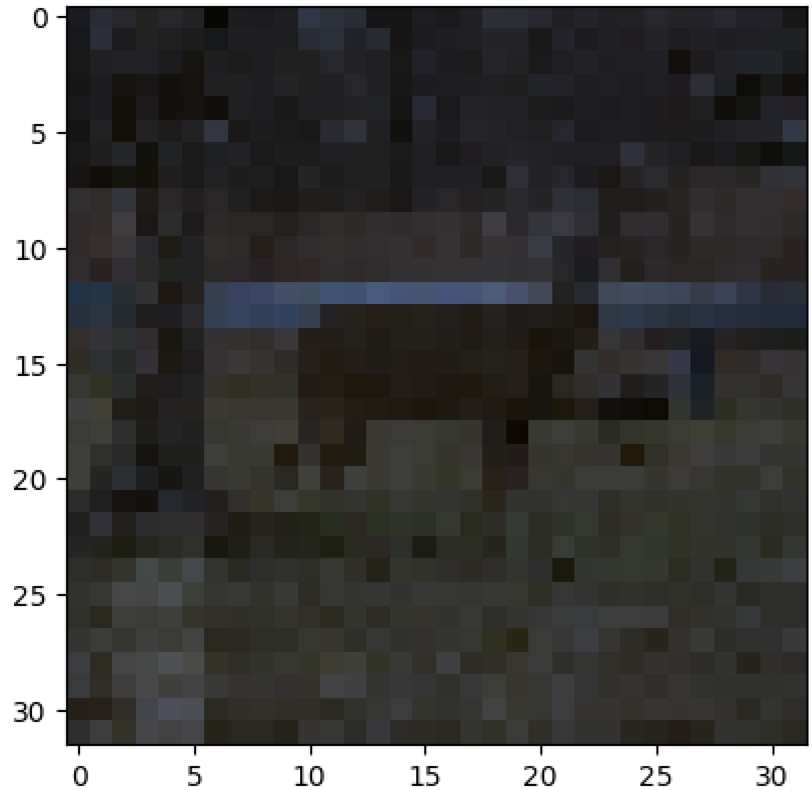

```
1/1 [==============================] - 0s 24ms/step
picture shows:     deer
model prediction:  airplane
wrong
(32, 32, 3)
```



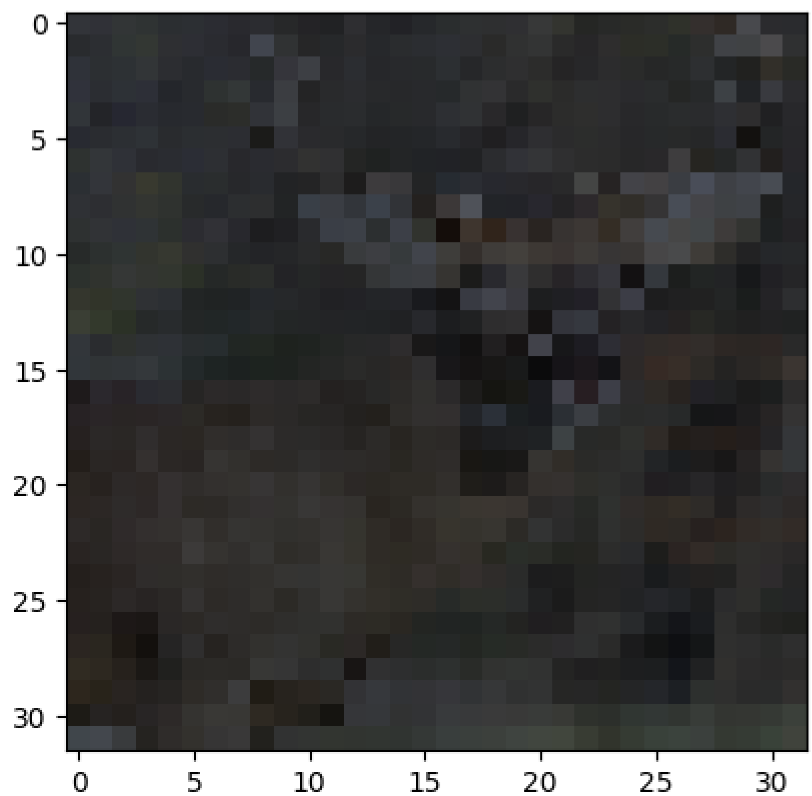
```
1/1 [==============================] - 0s 22ms/step
picture shows:     deer
model prediction:  deer
correct
(32, 32, 3)
```



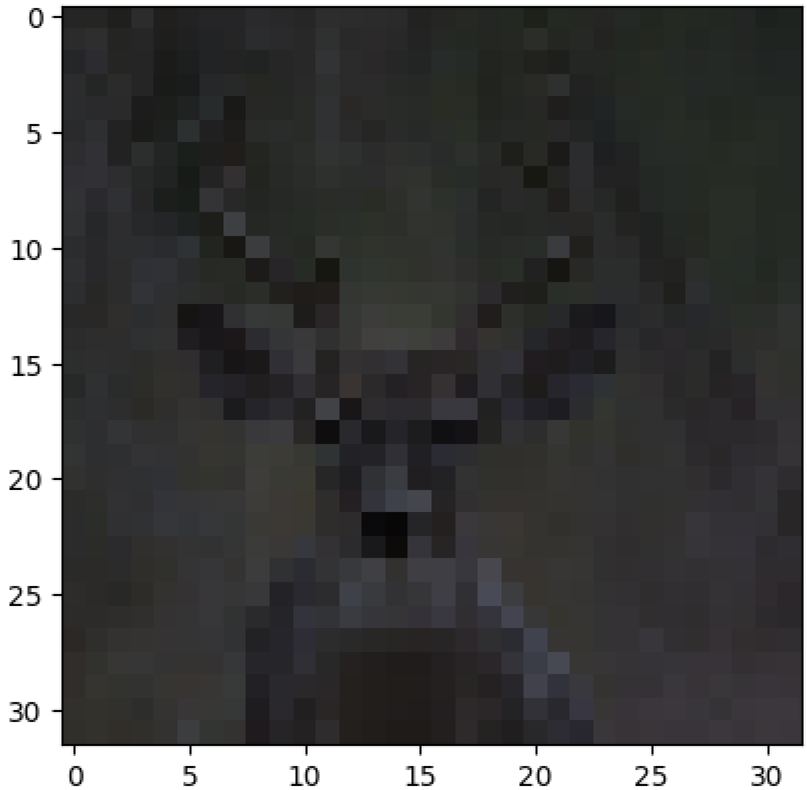
```
1/1 [==============================] - 0s 21ms/step
picture shows:     deer
model prediction:  dog
wrong
(32, 32, 3)
```



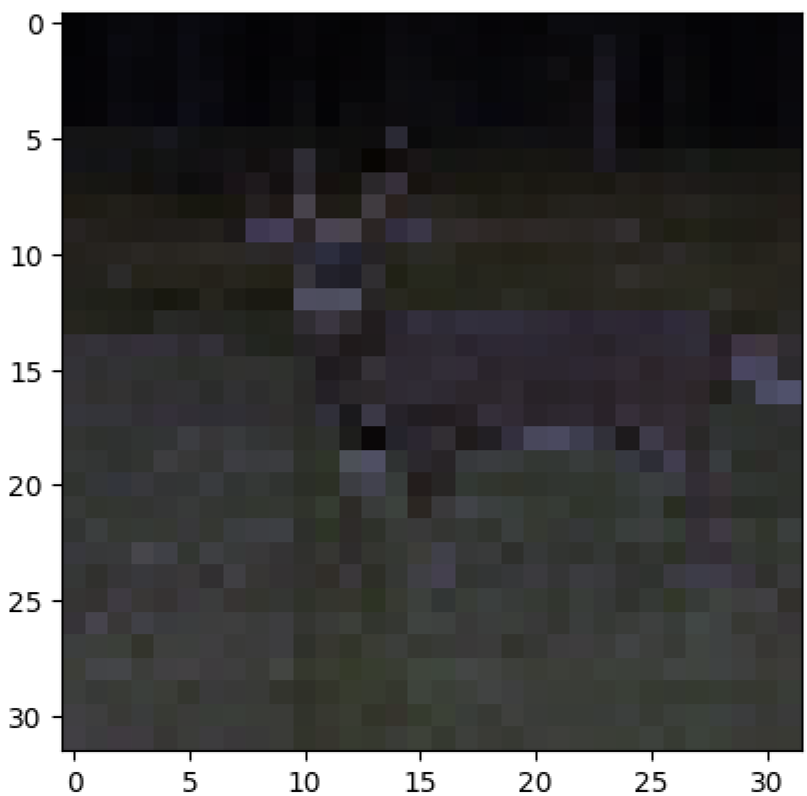
```
1/1 [==============================] - 0s 27ms/step
picture shows:     deer
model prediction:  ship
wrong
(32, 32, 3)
```



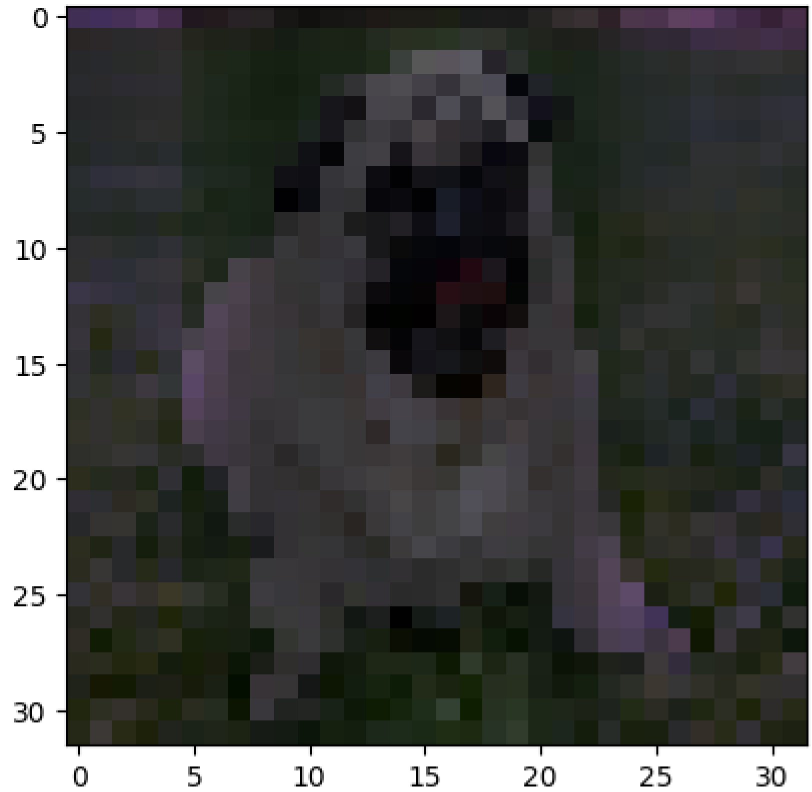
```
1/1 [==============================] - 0s 22ms/step
picture shows:     deer
model prediction:  deer
correct
(32, 32, 3)
```



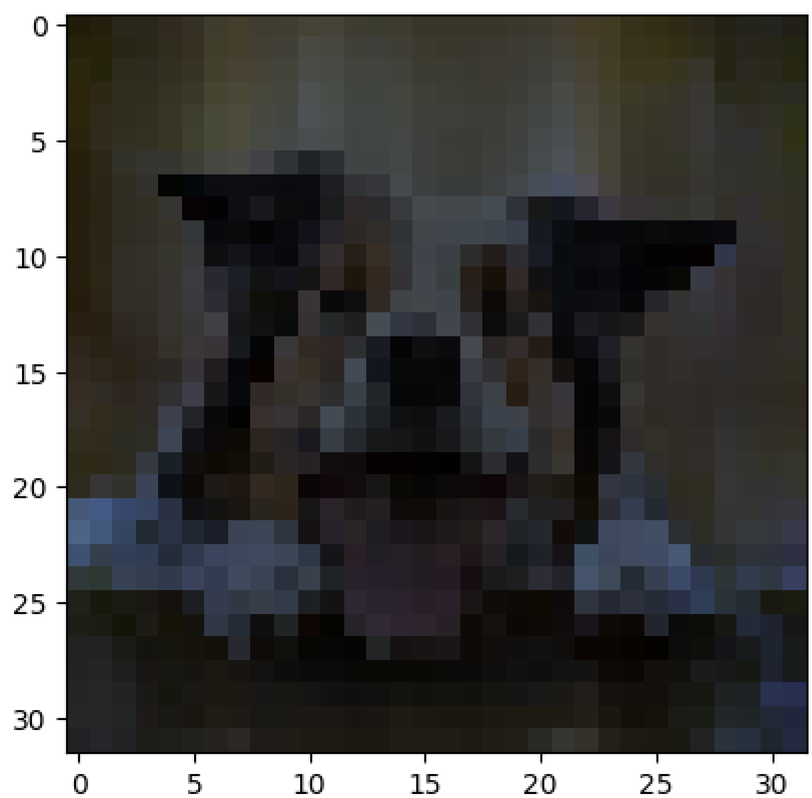
```
1/1 [==============================] - 0s 27ms/step
picture shows:     deer
model prediction:  deer
correct
(32, 32, 3)
```



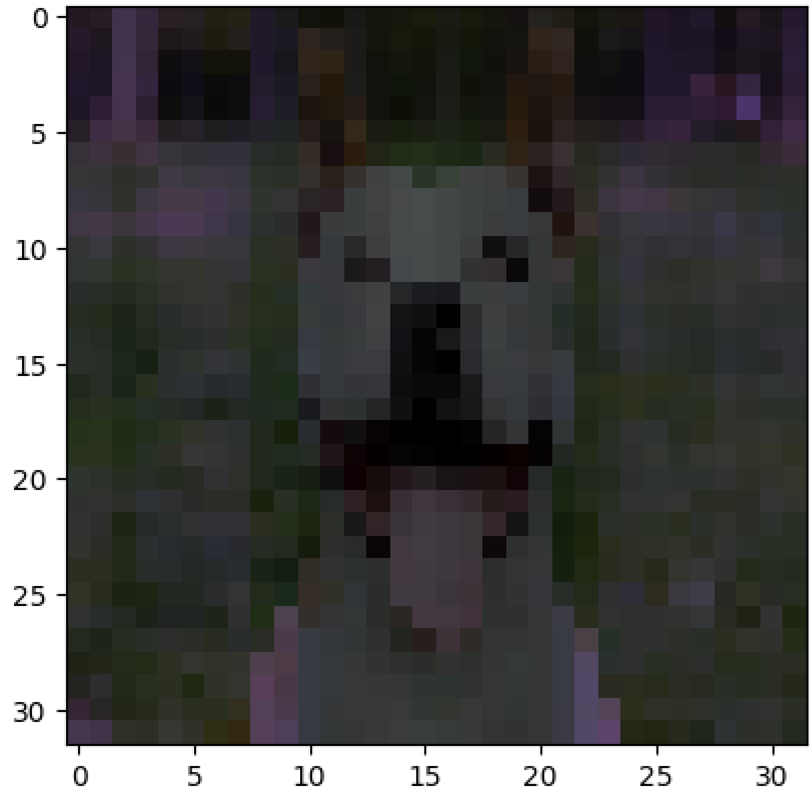
```
1/1 [==============================] - 0s 30ms/step
picture shows:     deer
model prediction:  deer
correct
(32, 32, 3)
```



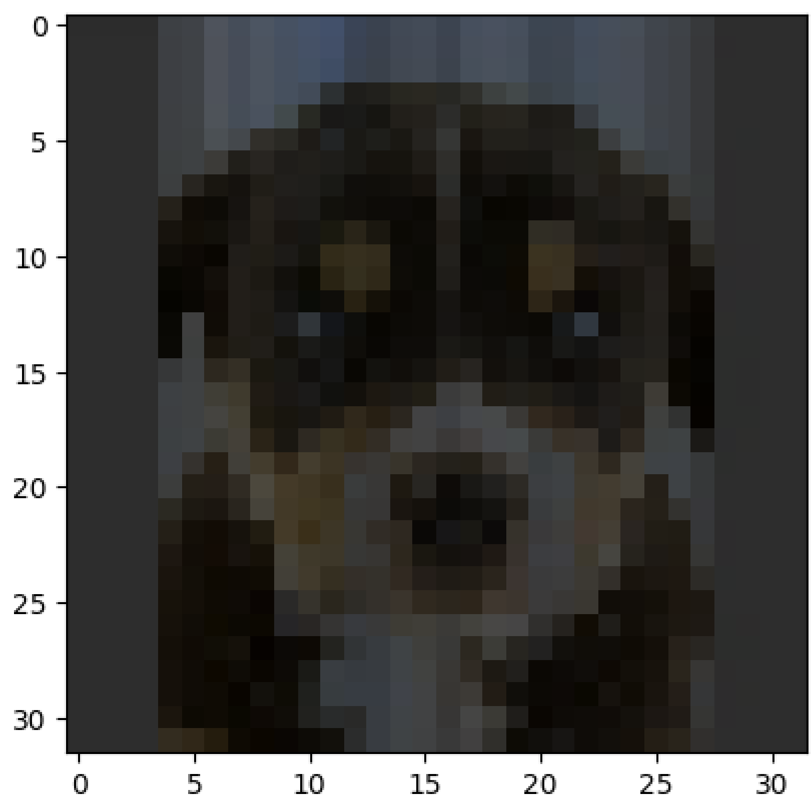
```
1/1 [==============================] - 0s 26ms/step
picture shows:     dog
model prediction:  dog
correct
(32, 32, 3)
```



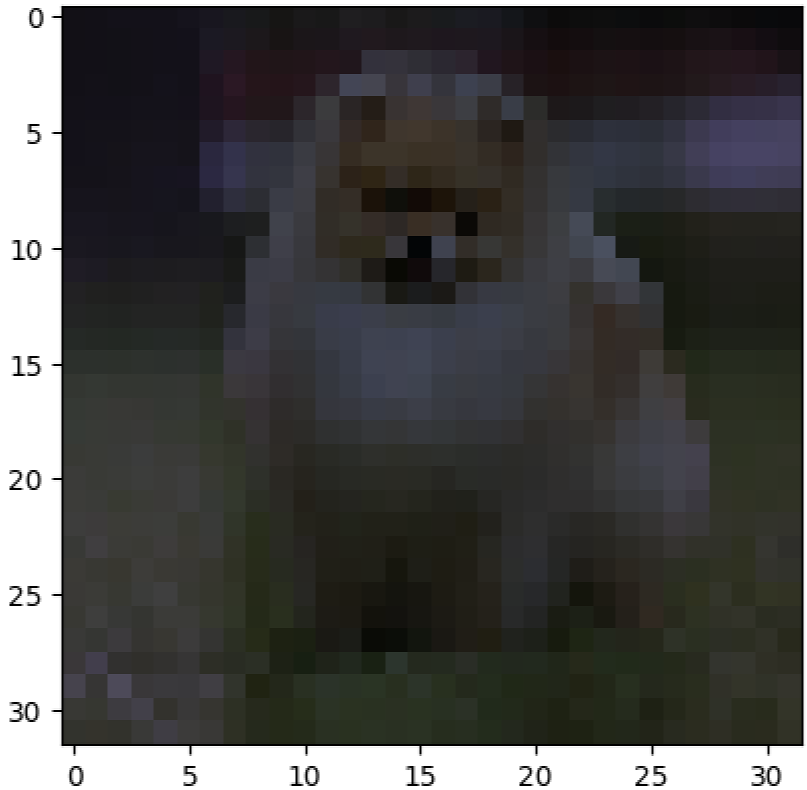
```
1/1 [==============================] - 0s 28ms/step
picture shows:     dog
model prediction:  dog
correct
(32, 32, 3)
```



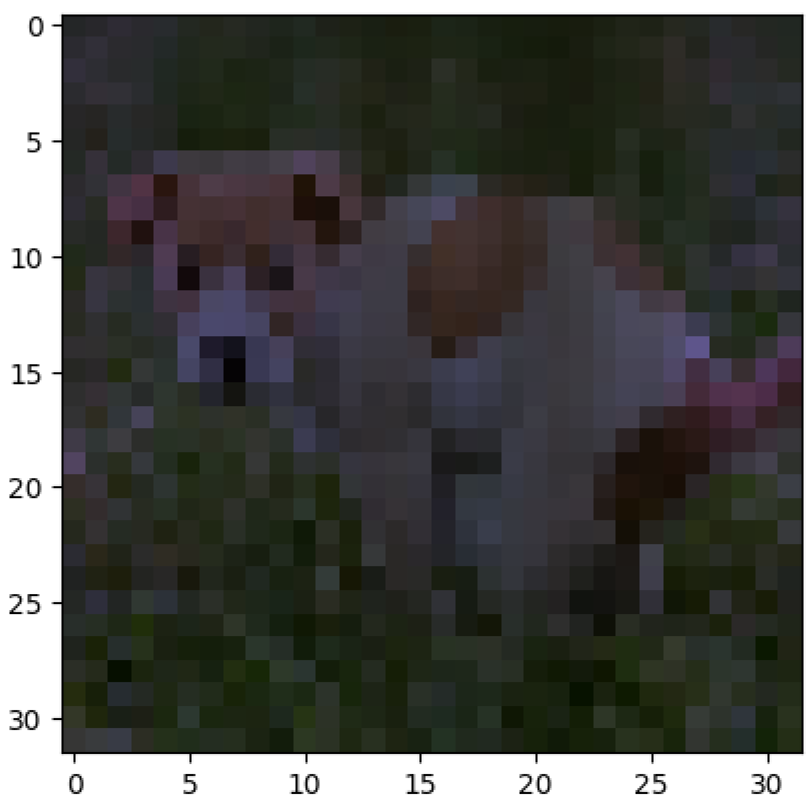

```
1/1 [==============================] - 0s 23ms/step
picture shows:     dog
model prediction:  bird
wrong
(32, 32, 3)
```



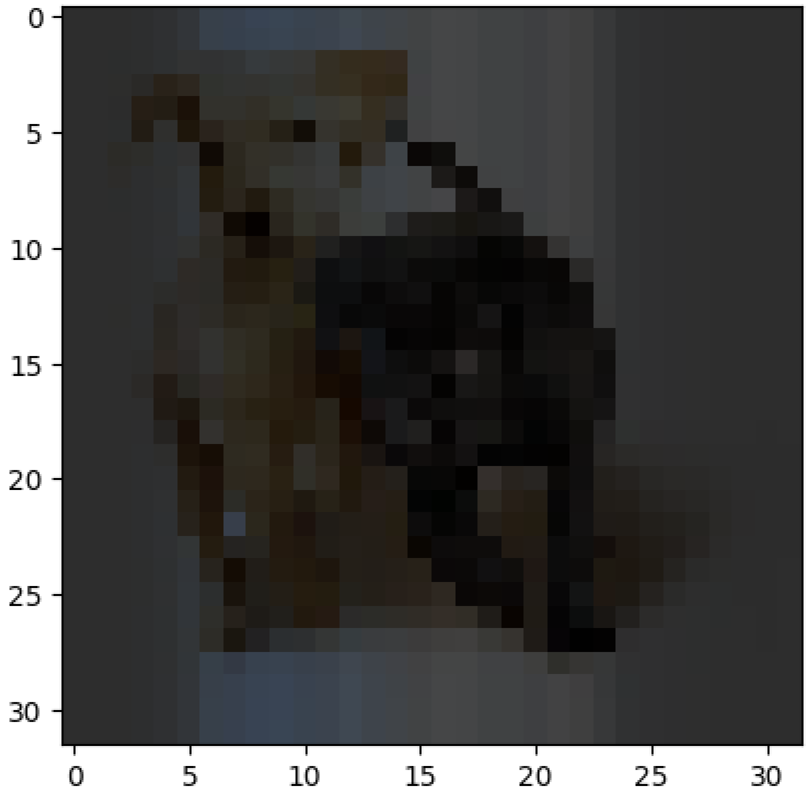

```
1/1 [==============================] - 0s 25ms/step
picture shows:     dog
model prediction:  dog
correct
(32, 32, 3)
```



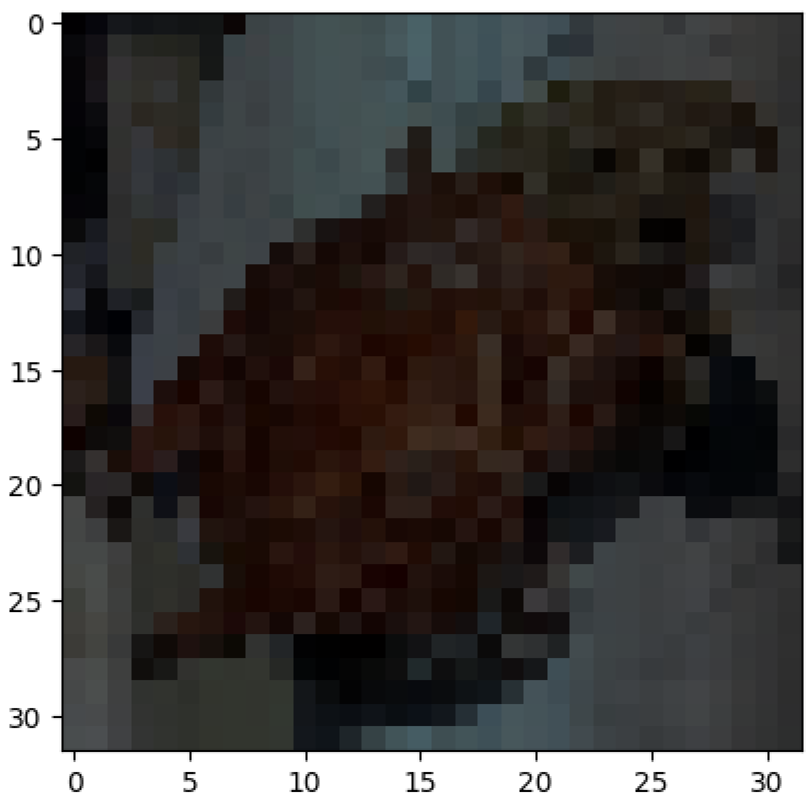

```
1/1 [==============================] - 0s 29ms/step
picture shows:     dog
model prediction:  dog
correct
(32, 32, 3)
```



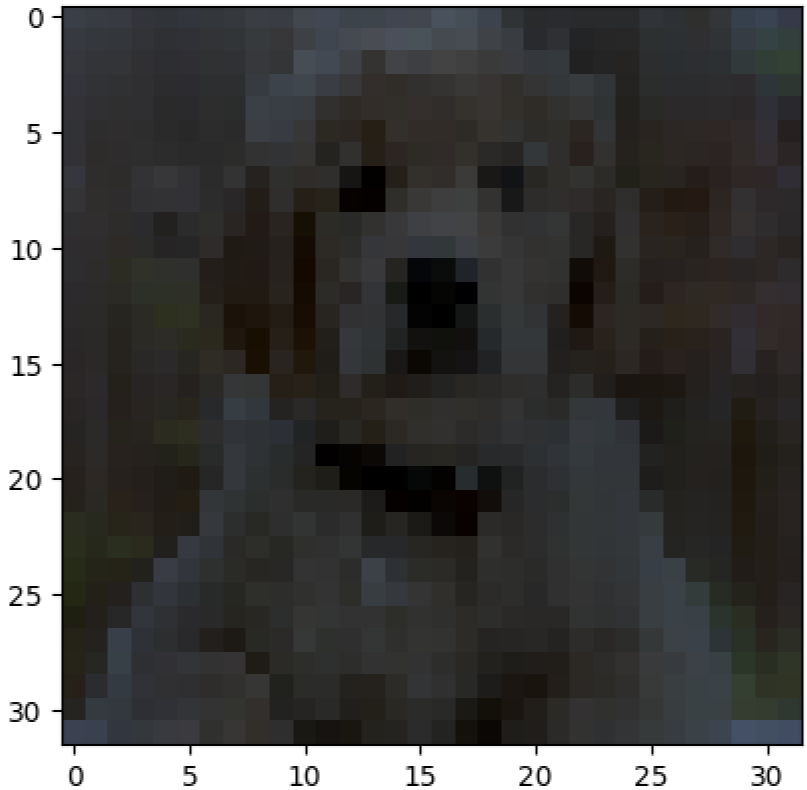

```
1/1 [==============================] - 0s 26ms/step
picture shows:     dog
model prediction:  frog
wrong
(32, 32, 3)
```



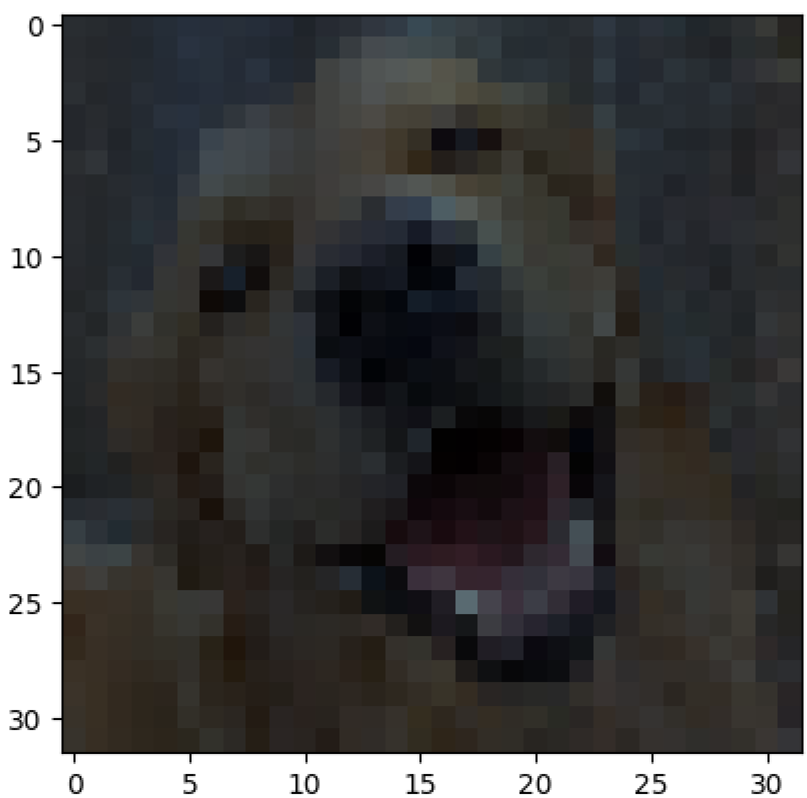

```
1/1 [==============================] - 0s 23ms/step
picture shows:     dog
model prediction:  cat
wrong
(32, 32, 3)
```



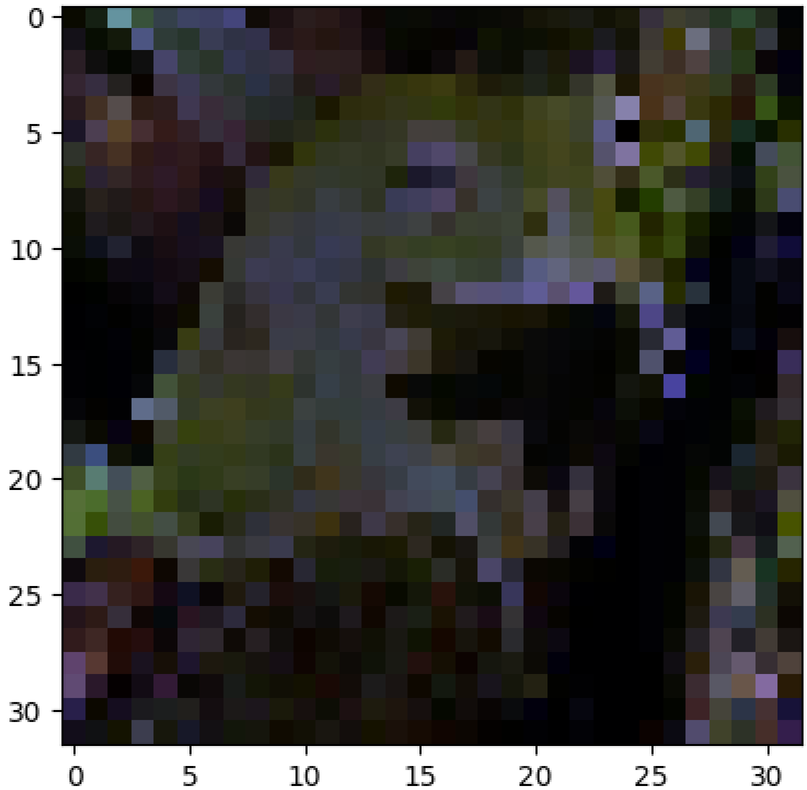

```
1/1 [==============================] - 0s 27ms/step
picture shows:     dog
model prediction:  truck
wrong
(32, 32, 3)
```



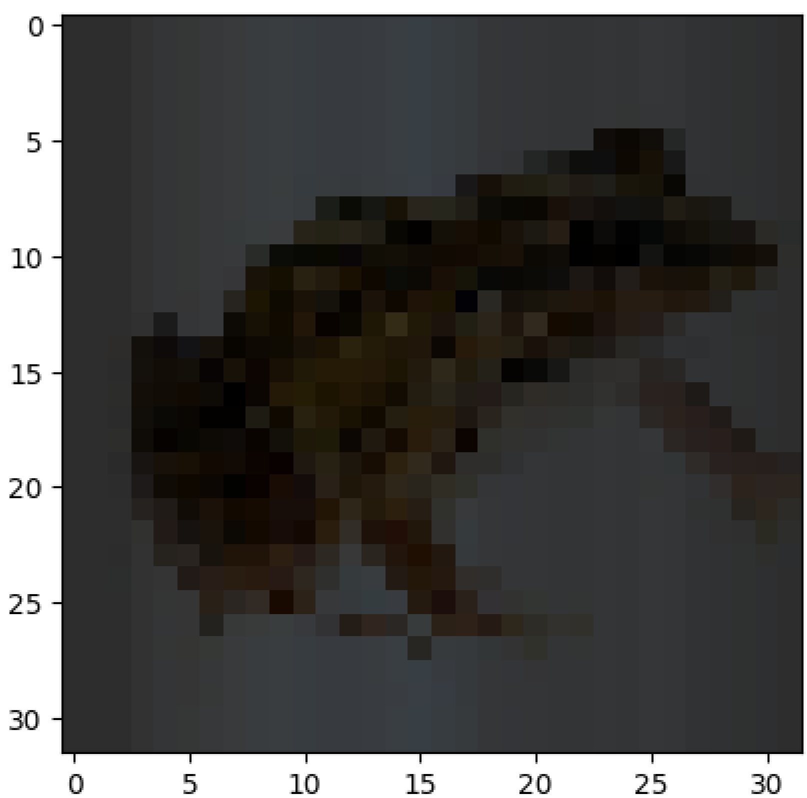

```
1/1 [==============================] - 0s 25ms/step
picture shows:     dog
model prediction:  dog
correct
(32, 32, 3)
```



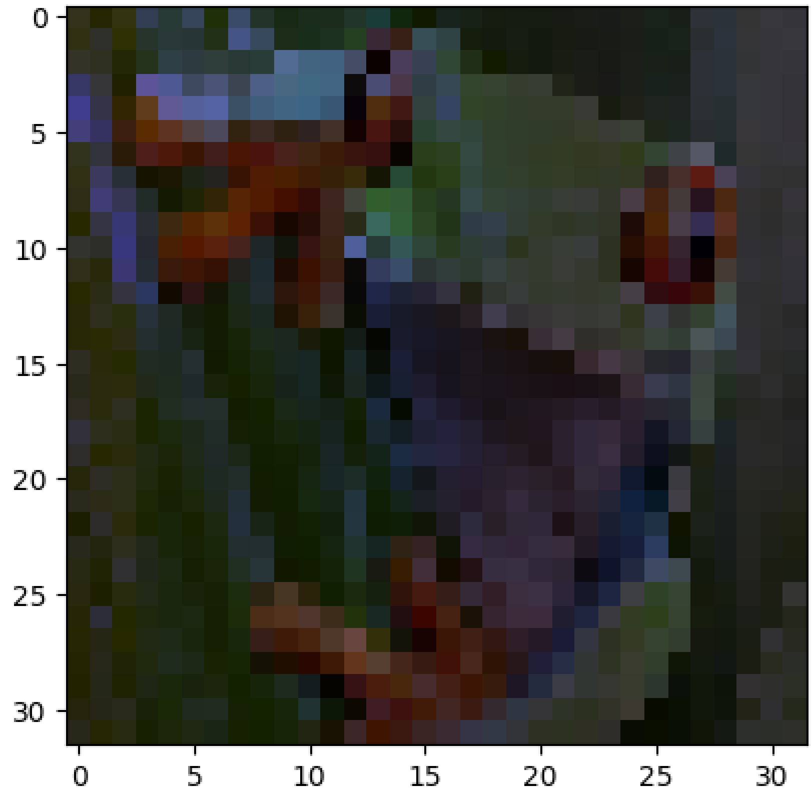

```
1/1 [==============================] - 0s 22ms/step
picture shows:     dog
model prediction:  cat
wrong
(32, 32, 3)
```



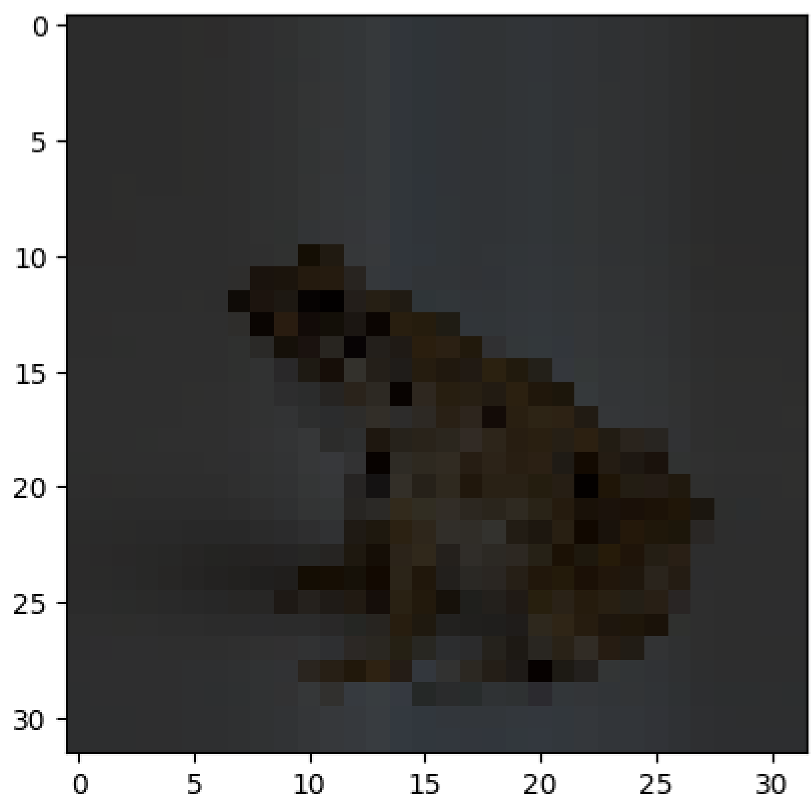

```
1/1 [==============================] - 0s 24ms/step
picture shows:     frog
model prediction:  frog
correct
(32, 32, 3)
```



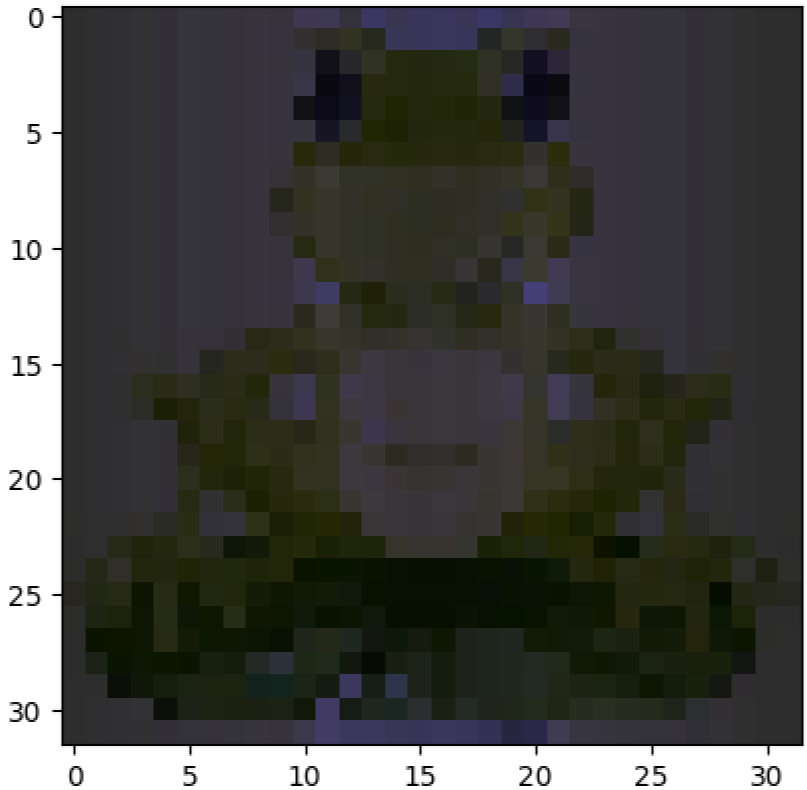

```
1/1 [==============================] - 0s 26ms/step
picture shows:     frog
model prediction:  frog
correct
(32, 32, 3)
```



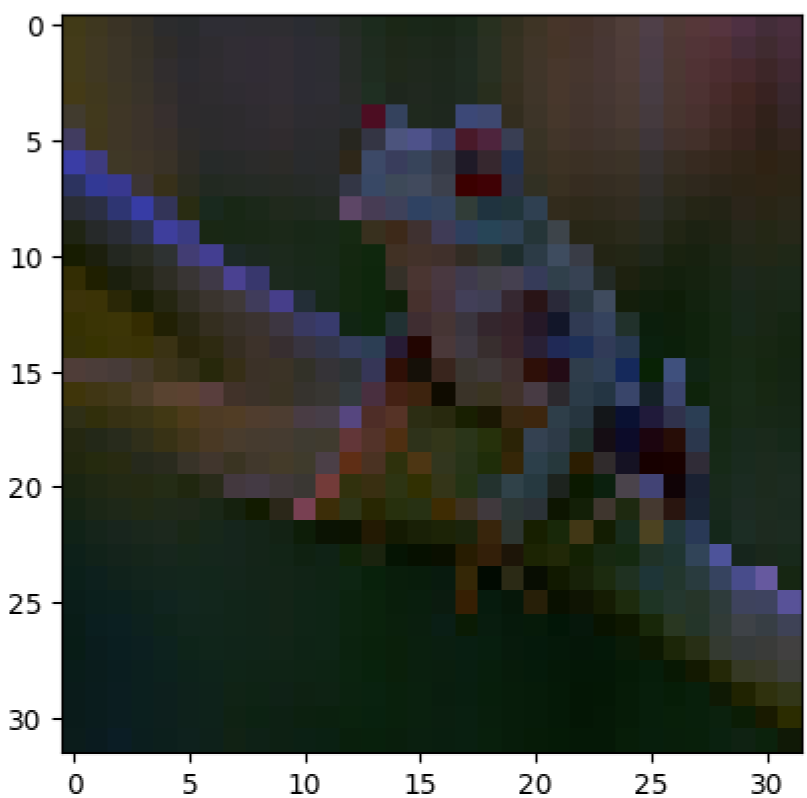

```
1/1 [==============================] - 0s 23ms/step
picture shows:      frog
model prediction:   cat
wrong
(32, 32, 3)
```



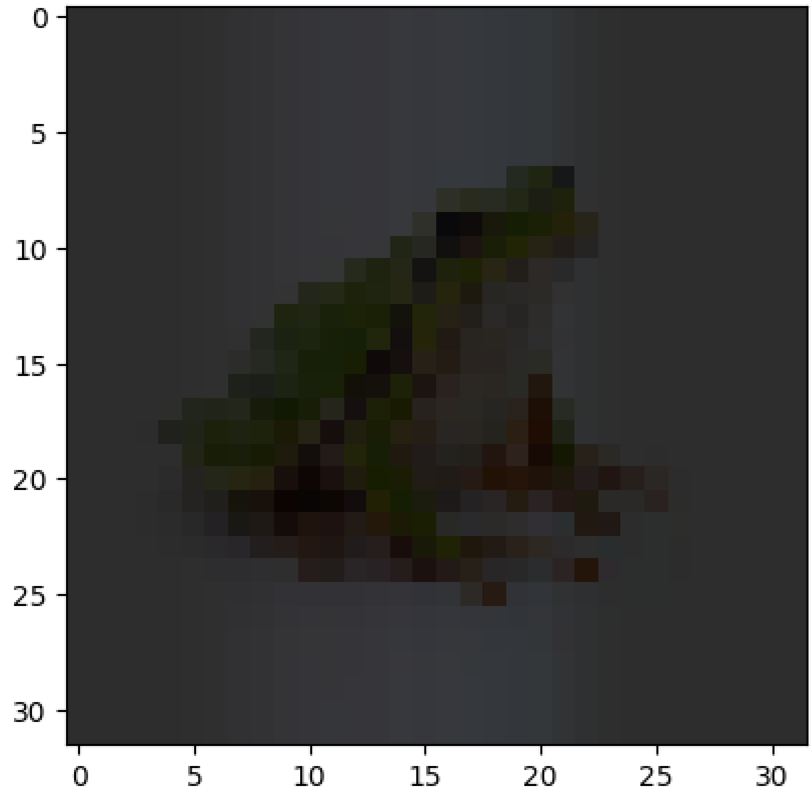

```
1/1 [==============================] - 0s 25ms/step
picture shows:      frog
model prediction:   frog
correct
(32, 32, 3)
```



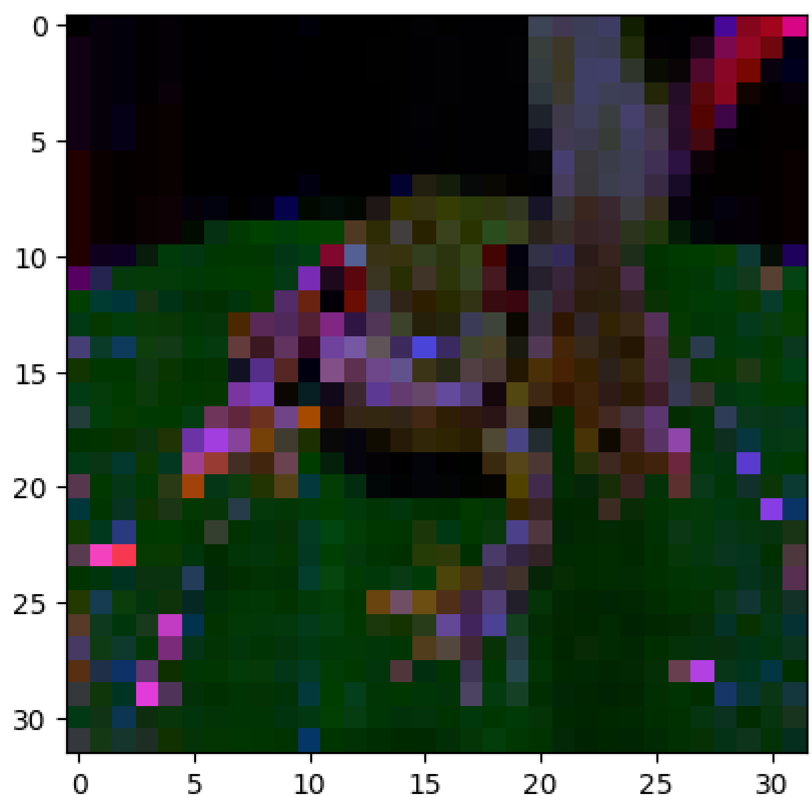
```
1/1 [==============================] - 0s 25ms/step
picture shows:      frog
model prediction:   frog
correct
(32, 32, 3)
```



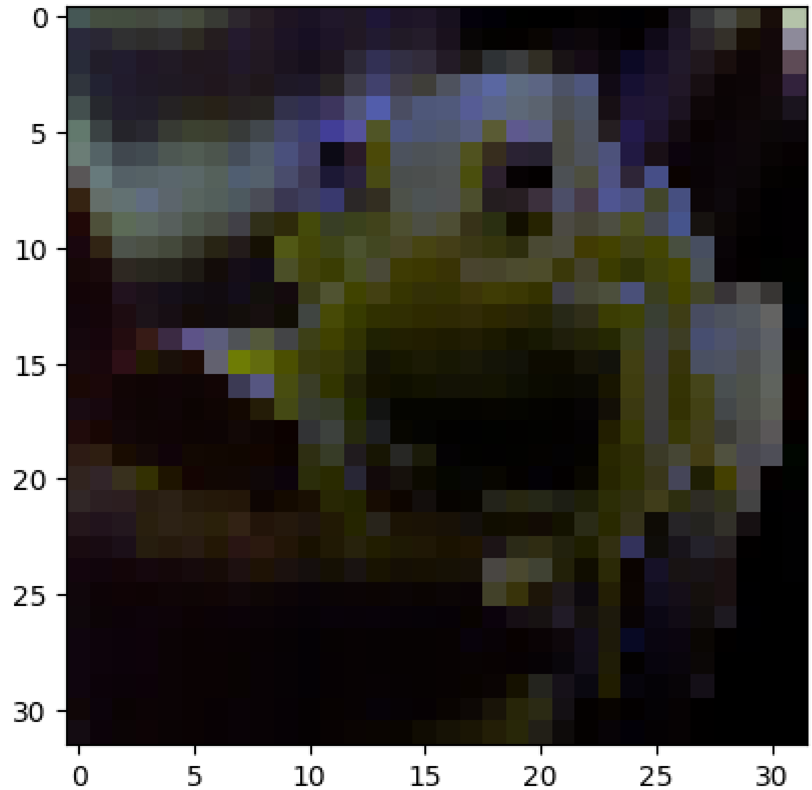
```
1/1 [==============================] - 0s 23ms/step
picture shows:      frog
model prediction:   frog
correct
(32, 32, 3)
```



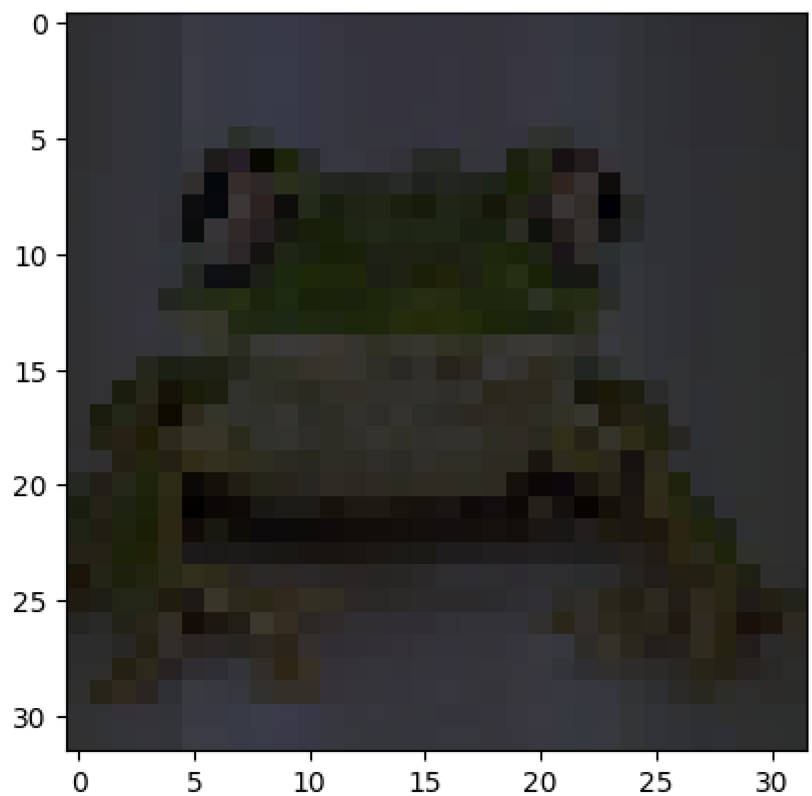
```
1/1 [==============================] - 0s 24ms/step
picture shows:      frog
model prediction:   frog
correct
(32, 32, 3)
```



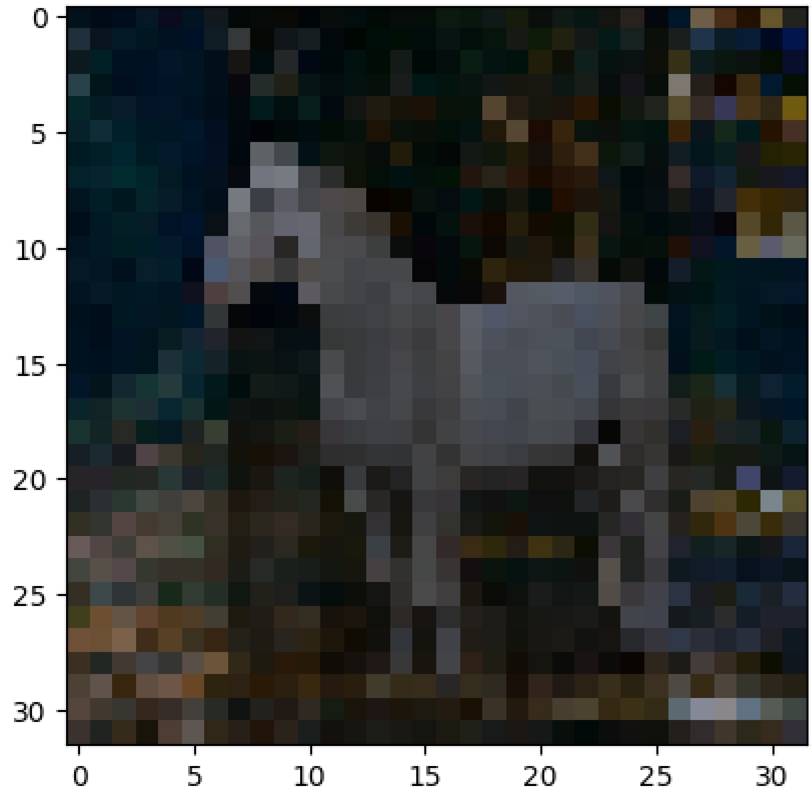
```
1/1 [==============================] - 0s 25ms/step
picture shows:      frog
model prediction:   frog
correct
(32, 32, 3)
```



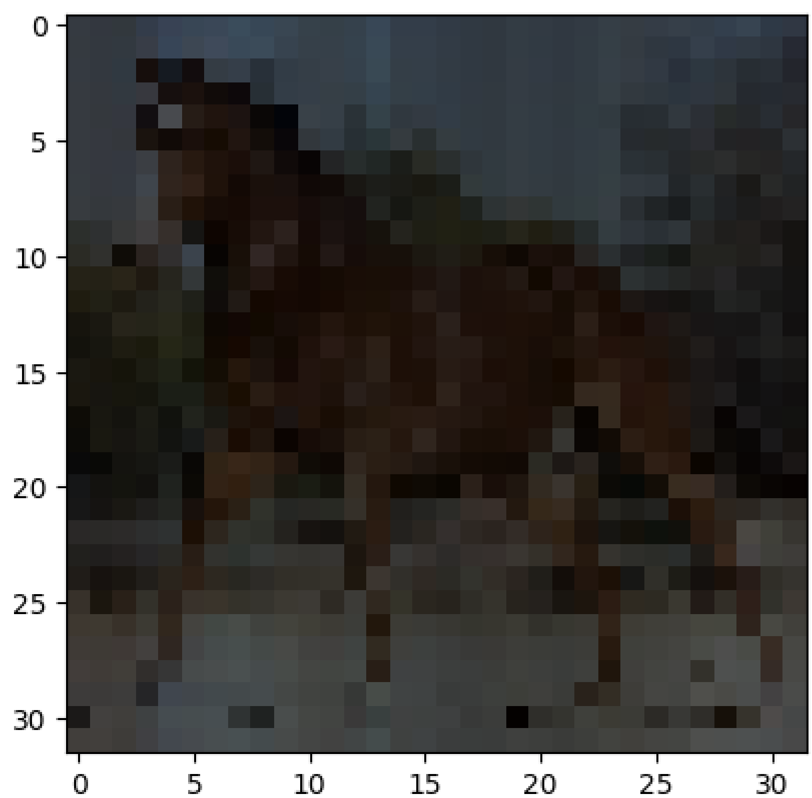
```
1/1 [==============================] - 0s 23ms/step
picture shows:      frog
model prediction:   bird
wrong
(32, 32, 3)
```



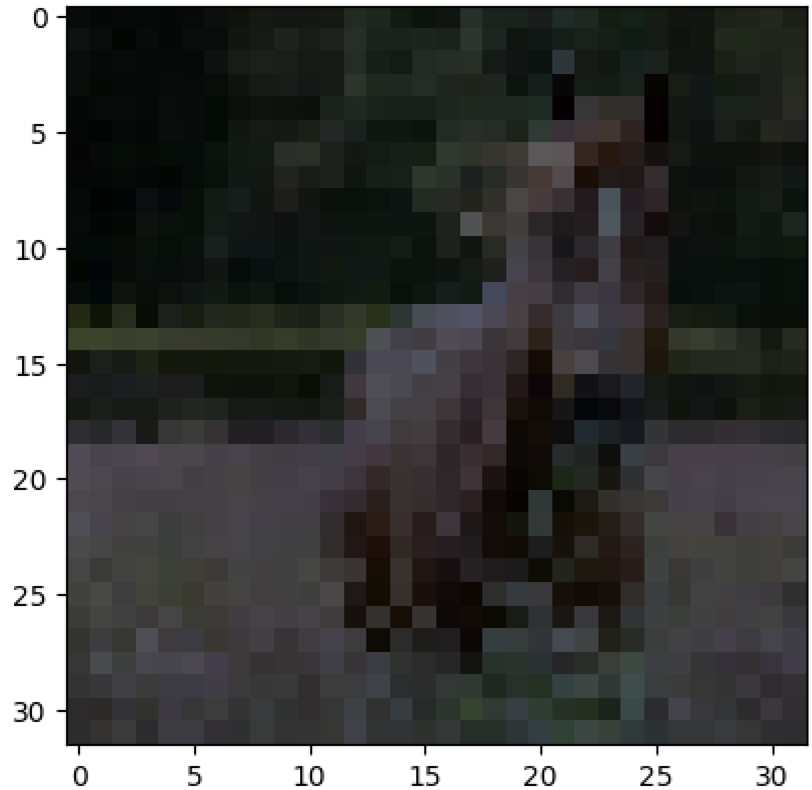
```
1/1 [==============================] - 0s 24ms/step
picture shows:      frog
model prediction:   frog
correct
(32, 32, 3)
```



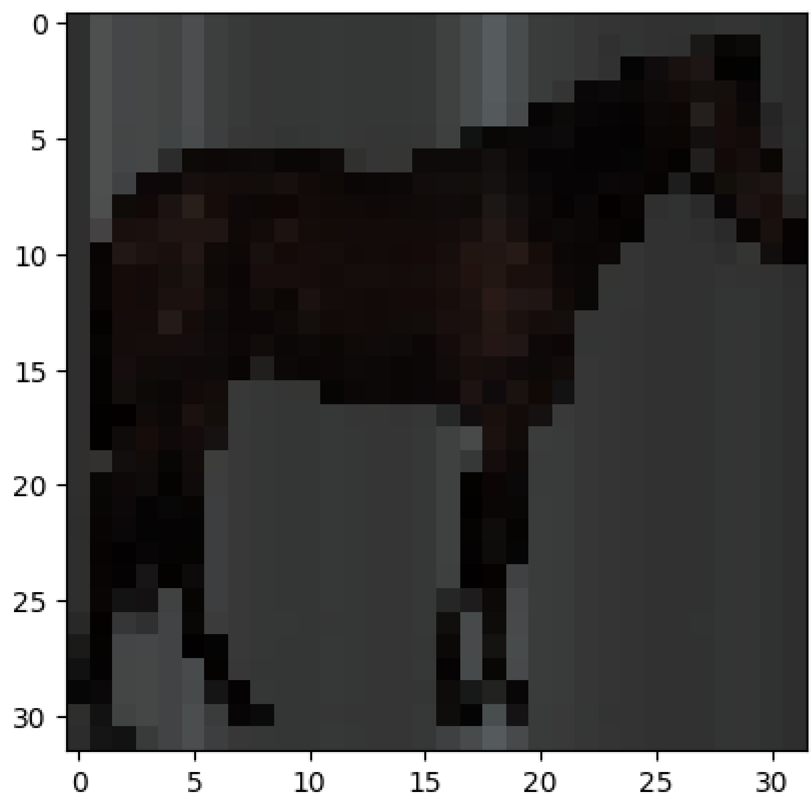

```
1/1 [==============================] - 0s 23ms/step
picture shows:      horse
model prediction:   horse
correct
(32, 32, 3)
```



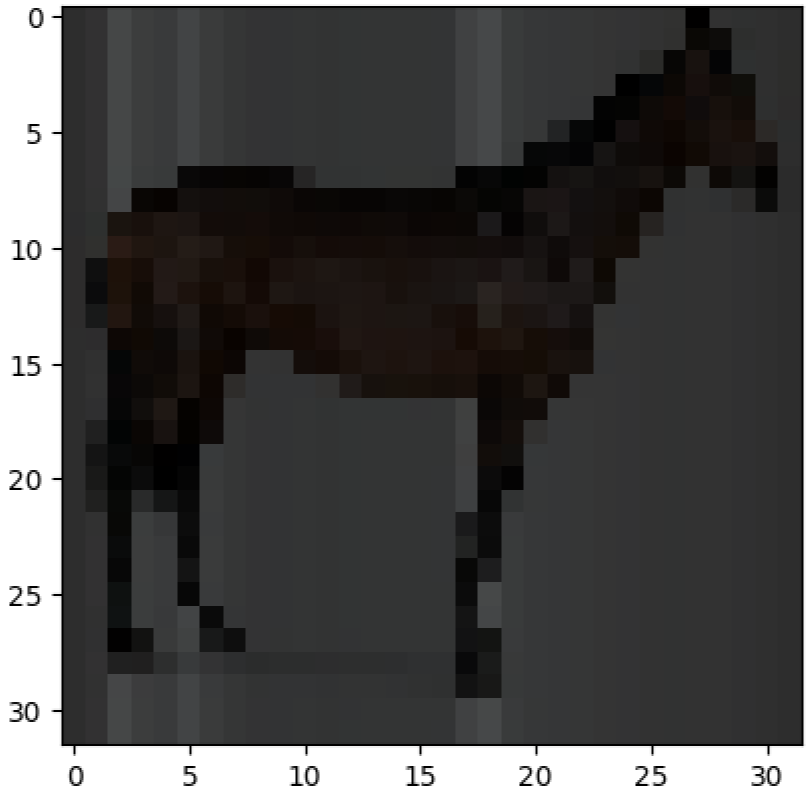

```
1/1 [==============================] - 0s 22ms/step
picture shows:      horse
model prediction:   horse
correct
(32, 32, 3)
```



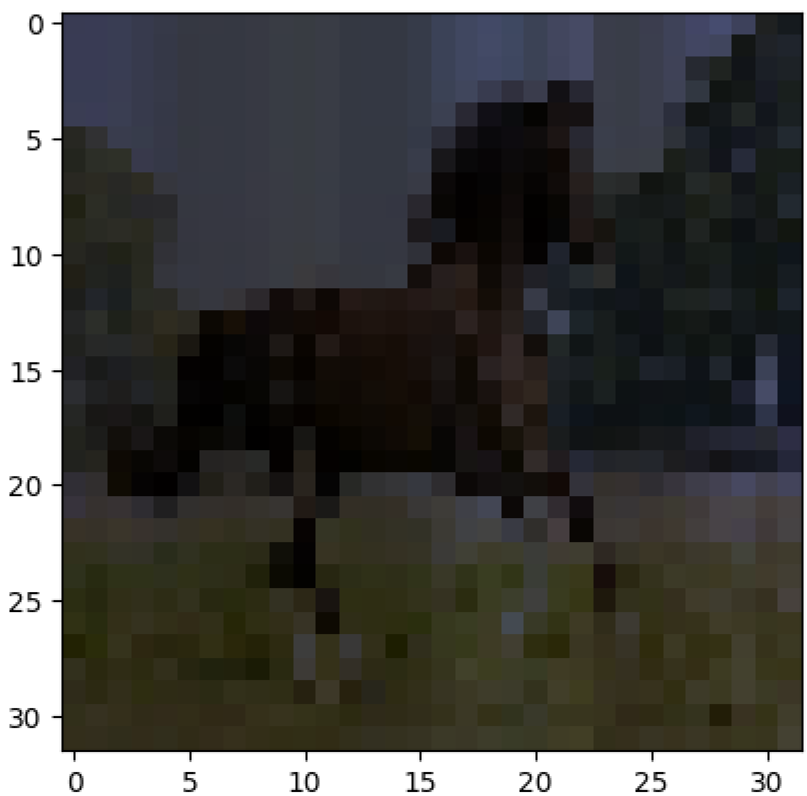

```
1/1 [==============================] - 0s 25ms/step
picture shows:     horse
model prediction:  deer
wrong
(32, 32, 3)
```



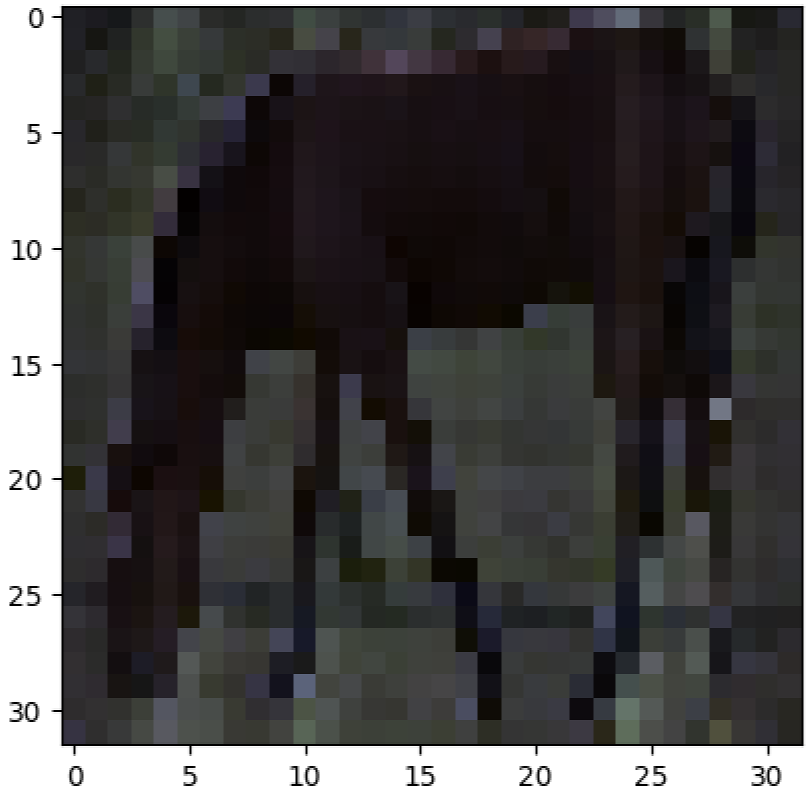

```
1/1 [==============================] - 0s 25ms/step
picture shows:     horse
model prediction:  horse
correct
(32, 32, 3)
```



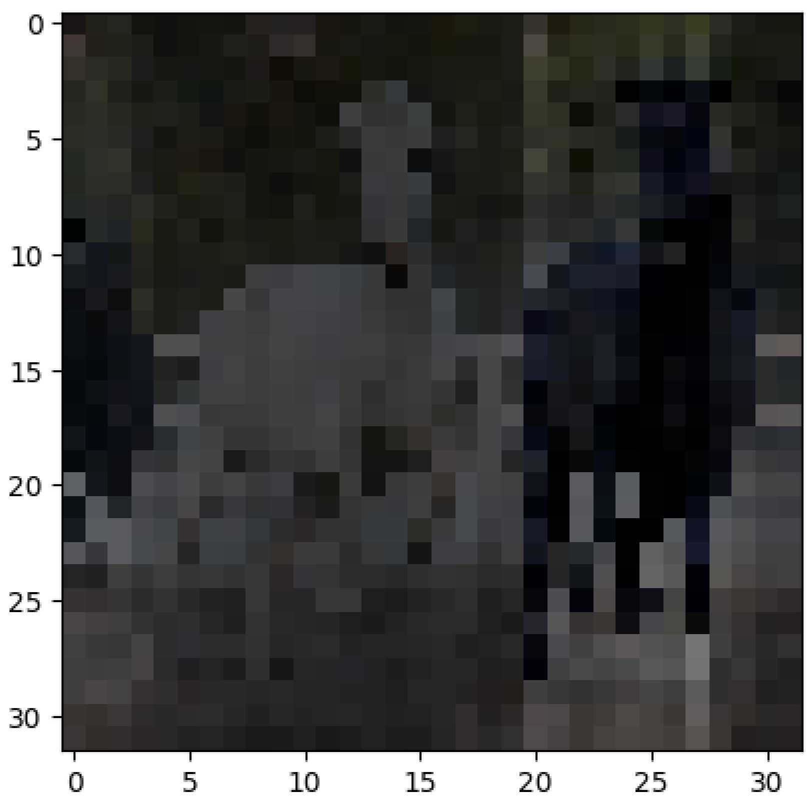
```
1/1 [==============================] - 0s 23ms/step
picture shows:     horse
model prediction:  horse
correct
(32, 32, 3)
```



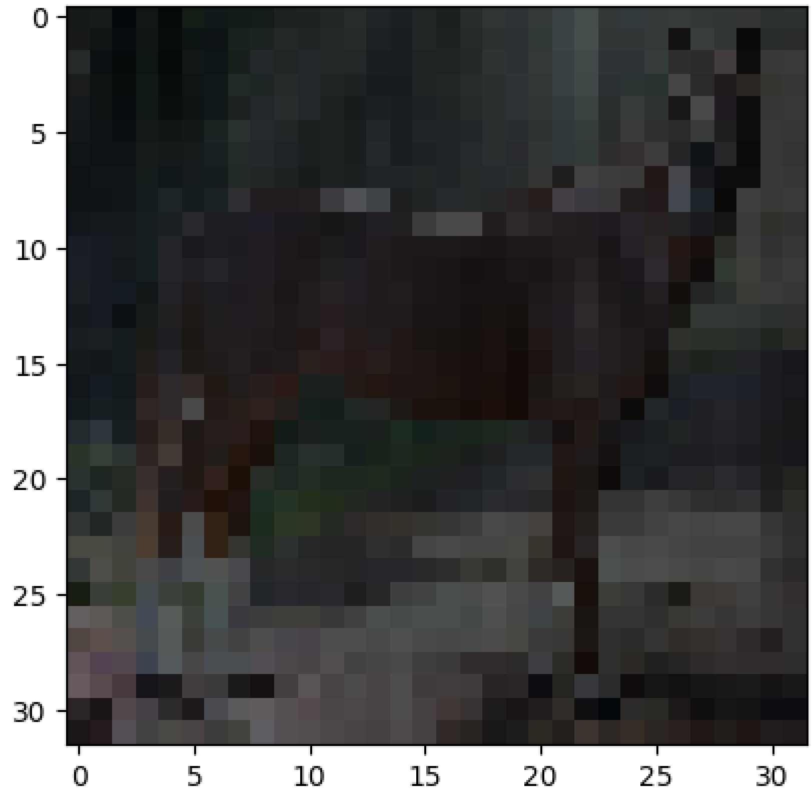
```
1/1 [==============================] - 0s 26ms/step
picture shows:     horse
model prediction:  horse
correct
(32, 32, 3)
```



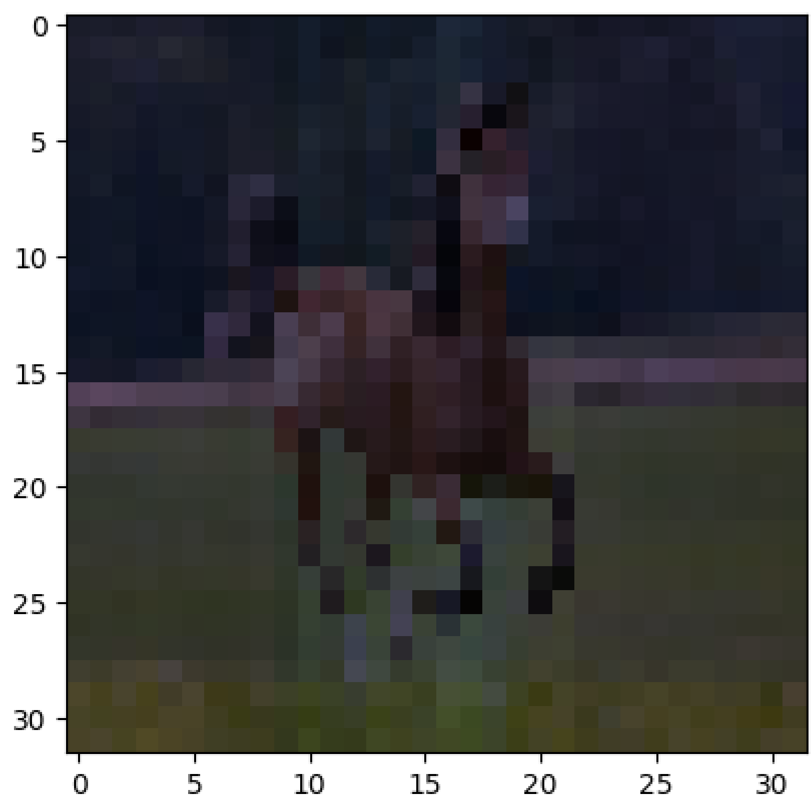

```
1/1 [==============================] - 0s 23ms/step
picture shows:     horse
model prediction:  horse
correct
(32, 32, 3)
```



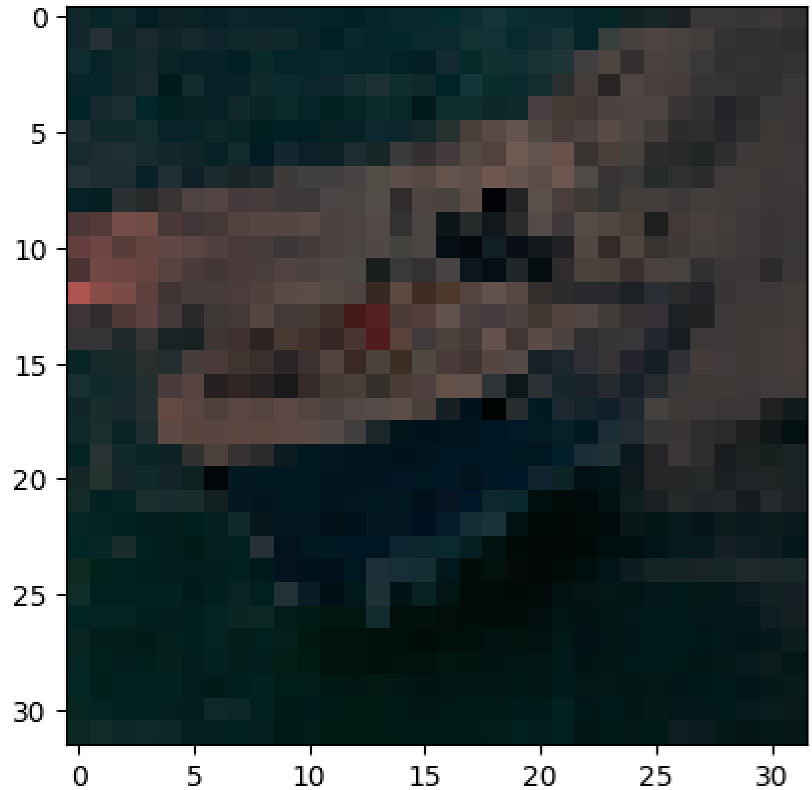

```
1/1 [==============================] - 0s 24ms/step
picture shows:     horse
model prediction:  horse
correct
(32, 32, 3)
```



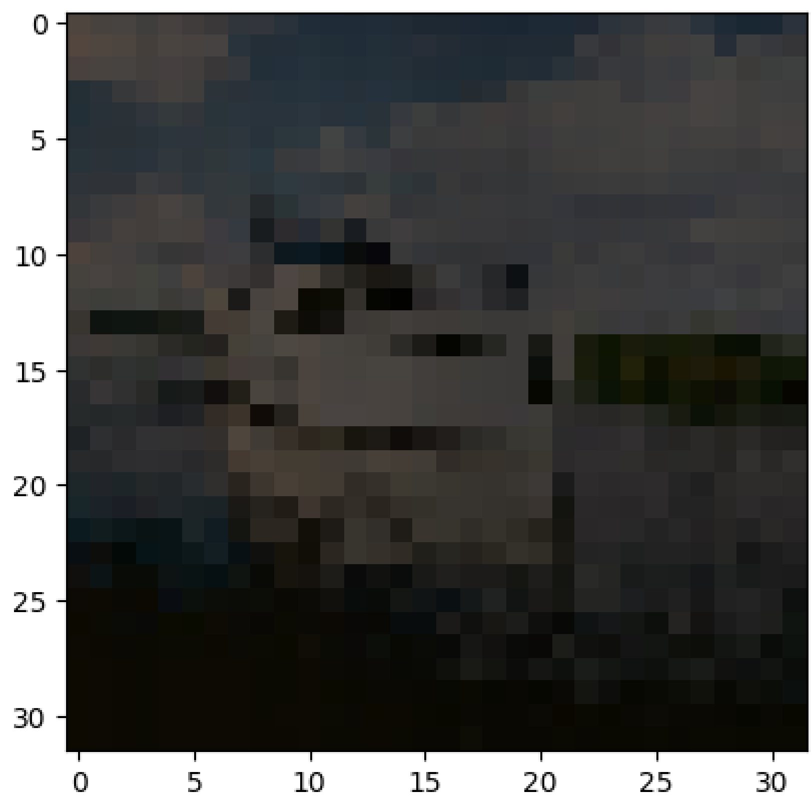

```
1/1 [==============================] - 0s 26ms/step
picture shows:     horse
model prediction:  horse
correct
(32, 32, 3)
```



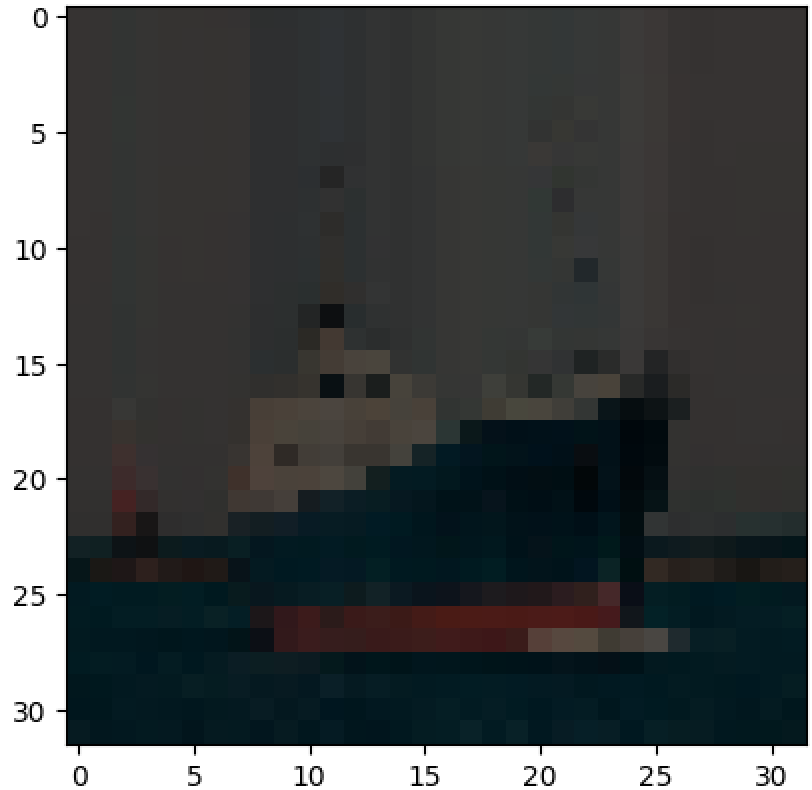

```
1/1 [==============================] - 0s 23ms/step
picture shows:     horse
model prediction:  horse
correct
(32, 32, 3)
```



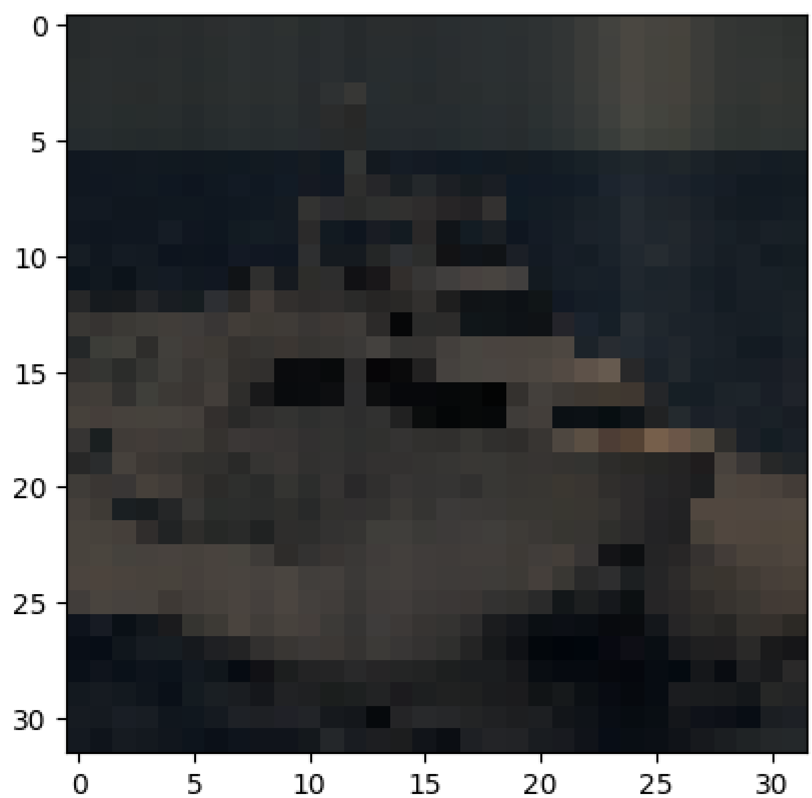

```
1/1 [==============================] - 0s 23ms/step
picture shows:     ship
model prediction:  ship
correct
(32, 32, 3)
```



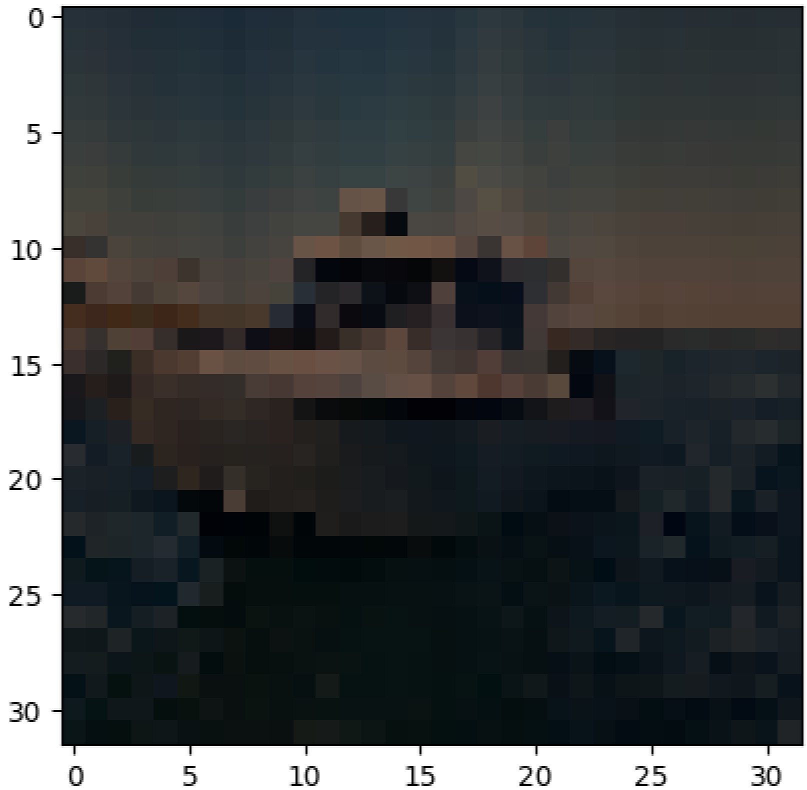

```
1/1 [==============================] - 0s 25ms/step
picture shows:     ship
model prediction:  ship
correct
(32, 32, 3)
```



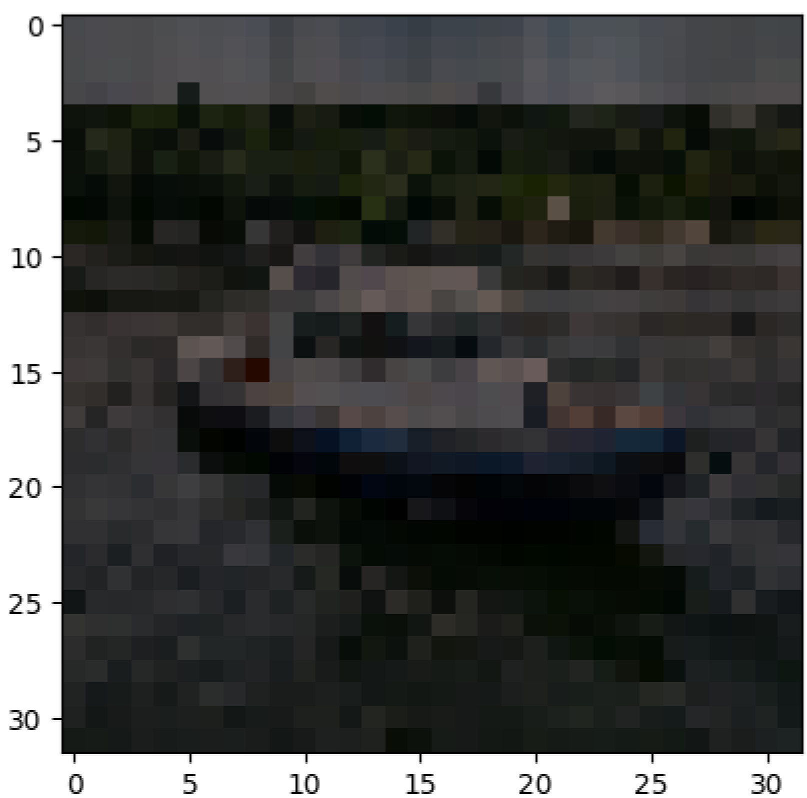
```
1/1 [==============================] - 0s 27ms/step
picture shows:     ship
model prediction:  ship
correct
(32, 32, 3)
```



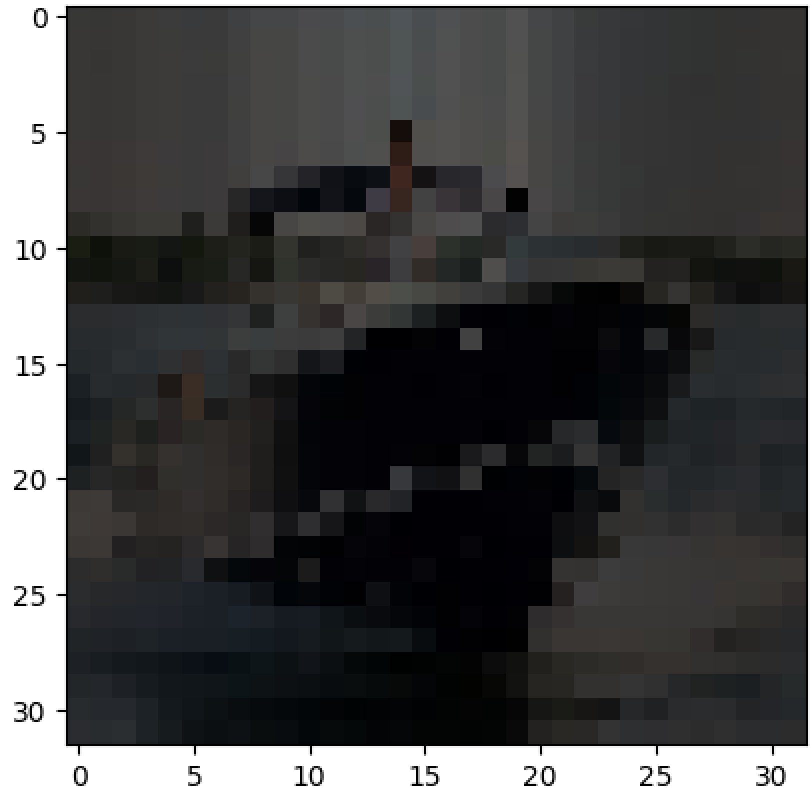
```
1/1 [==============================] - 0s 26ms/step
picture shows:     ship
model prediction:  ship
correct
(32, 32, 3)
```



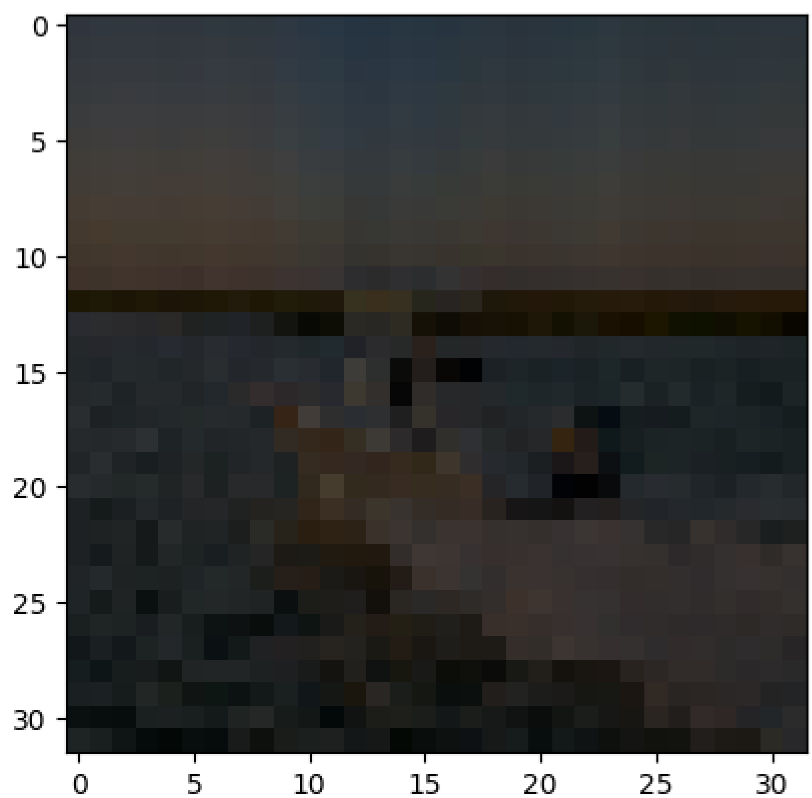
```
1/1 [==============================] - 0s 24ms/step
picture shows:     ship
model prediction:  ship
correct
(32, 32, 3)
```



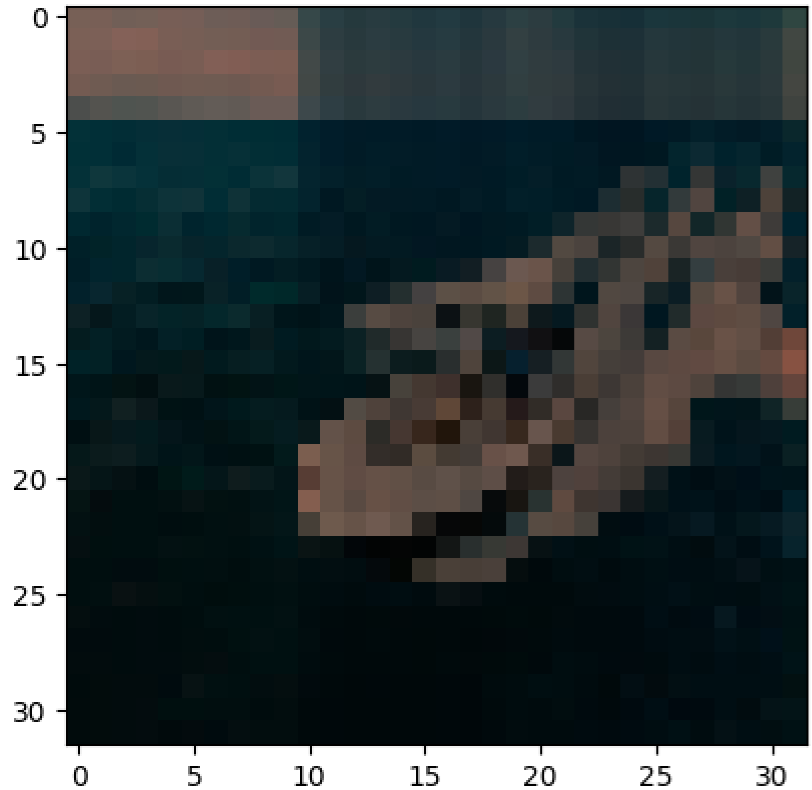
```
1/1 [==============================] - 0s 20ms/step
picture shows:     ship
model prediction:  ship
correct
(32, 32, 3)
```



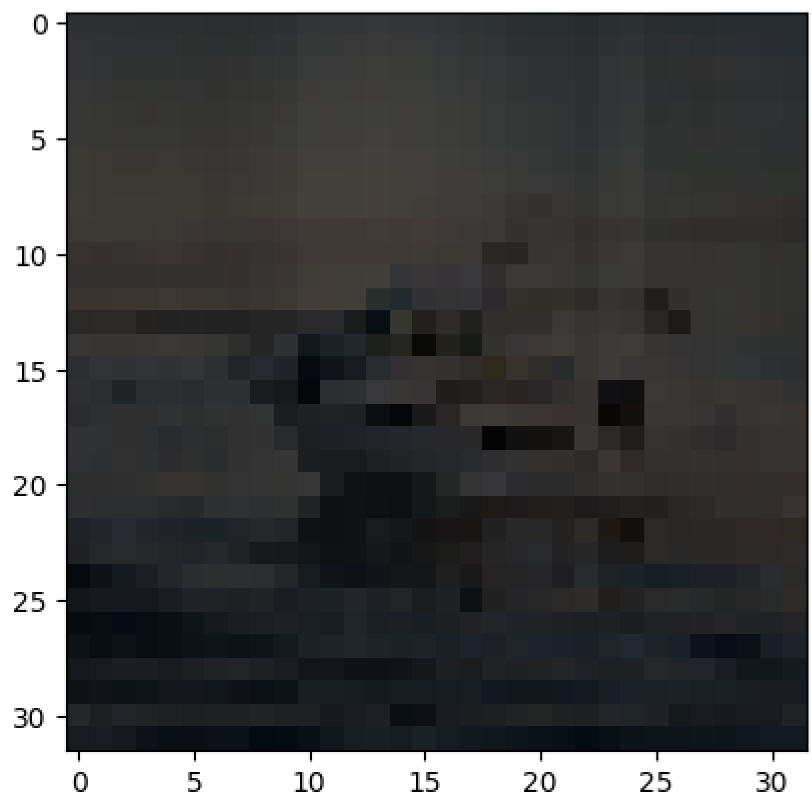

```
1/1 [==============================] - 0s 24ms/step
picture shows:     ship
model prediction:  airplane
wrong
(32, 32, 3)
```



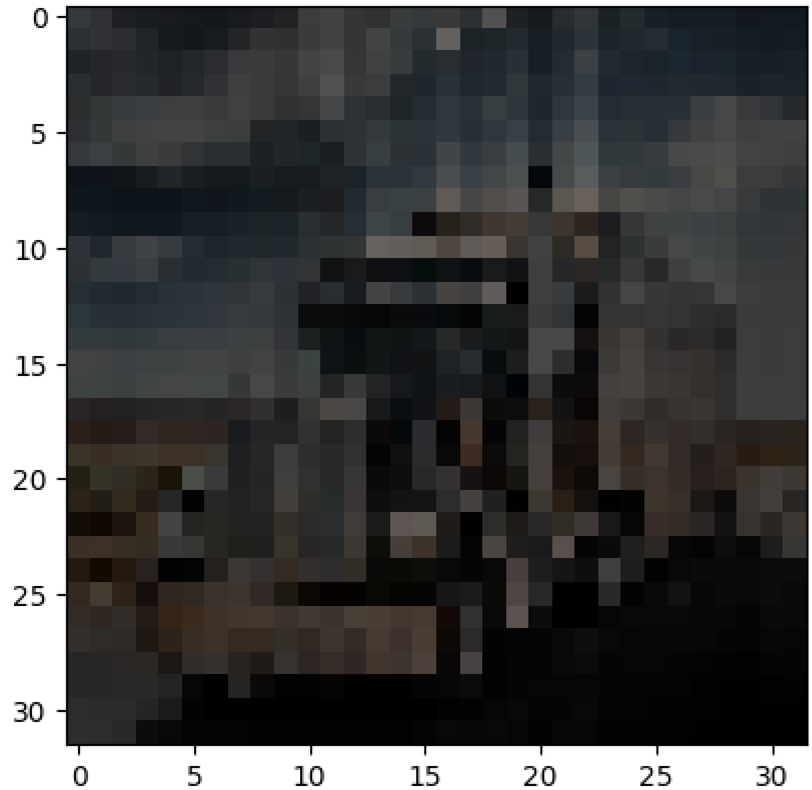

```
1/1 [==============================] - 0s 26ms/step
picture shows:     ship
model prediction:  ship
correct
(32, 32, 3)
```



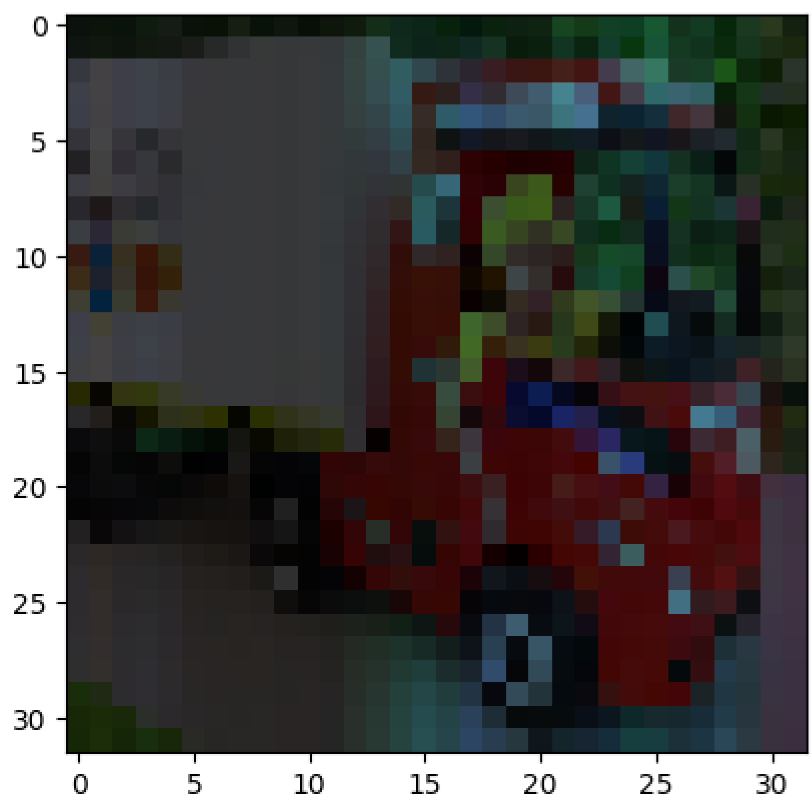
```
1/1 [==============================] - 0s 28ms/step
picture shows:     ship
model prediction:  ship
correct
(32, 32, 3)
```



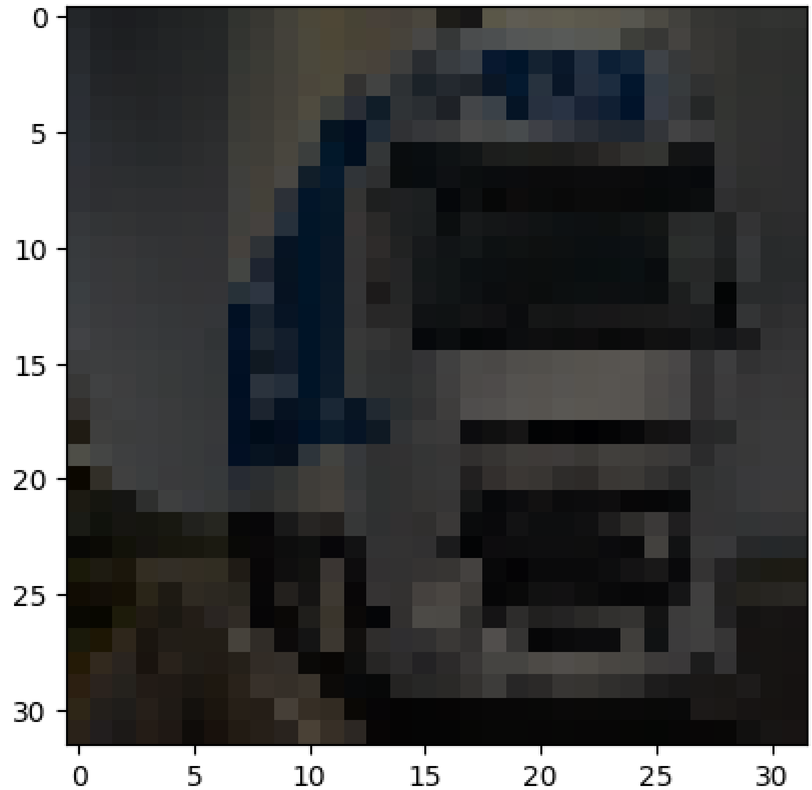
```
1/1 [==============================] - 0s 22ms/step
picture shows:     ship
model prediction:  airplane
wrong
(32, 32, 3)
```



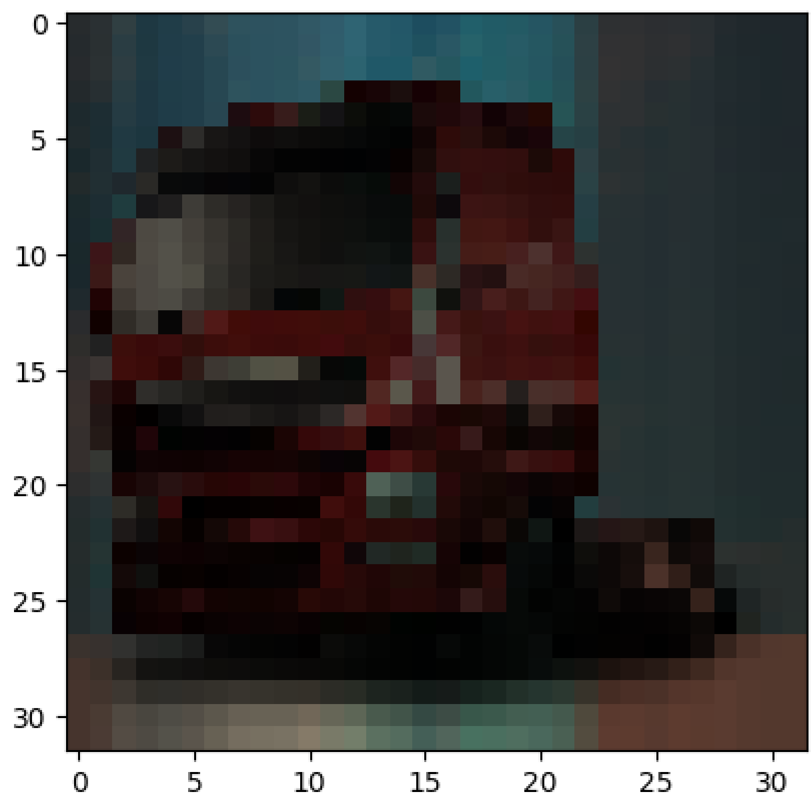
```
1/1 [==============================] - 0s 22ms/step
picture shows:      truck
model prediction:   truck
correct
(32, 32, 3)
```



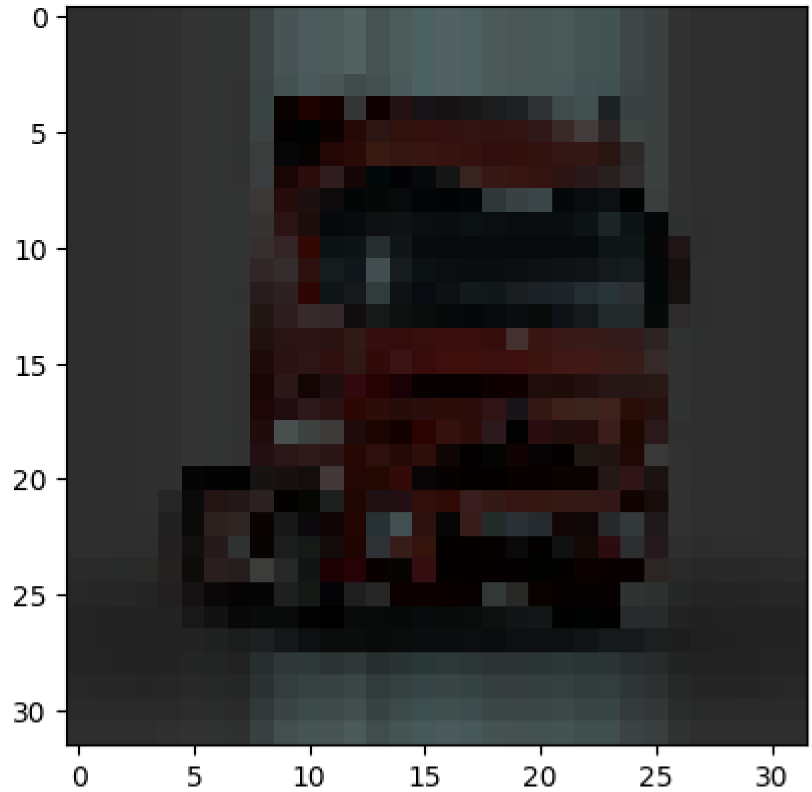
```
1/1 [==============================] - 0s 30ms/step
picture shows:      truck
model prediction:   truck
correct
(32, 32, 3)
```



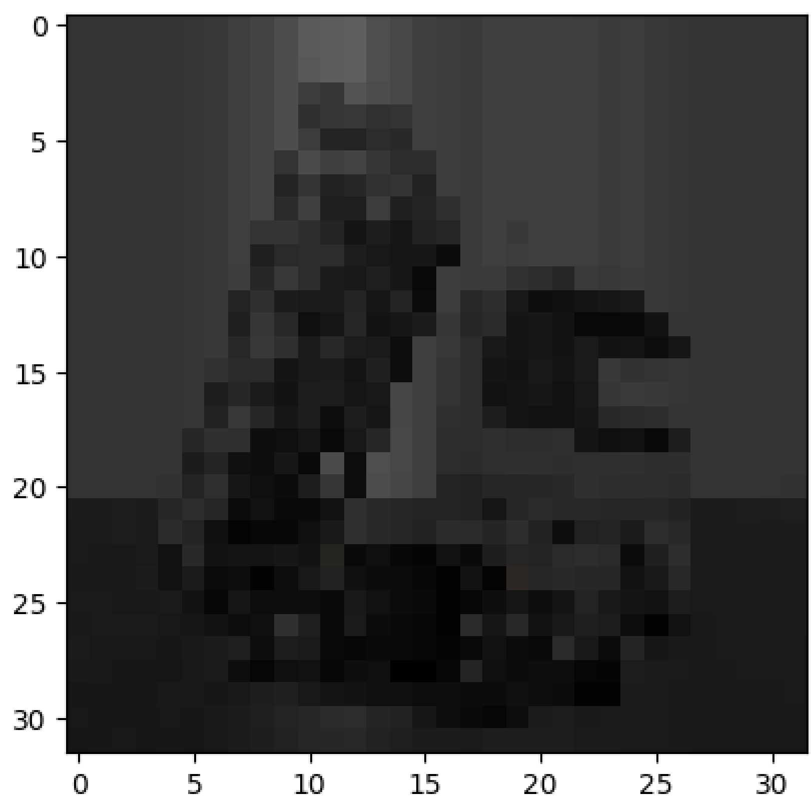

```
1/1 [==============================] - 0s 25ms/step
picture shows:      truck
model prediction:   truck
correct
(32, 32, 3)
```



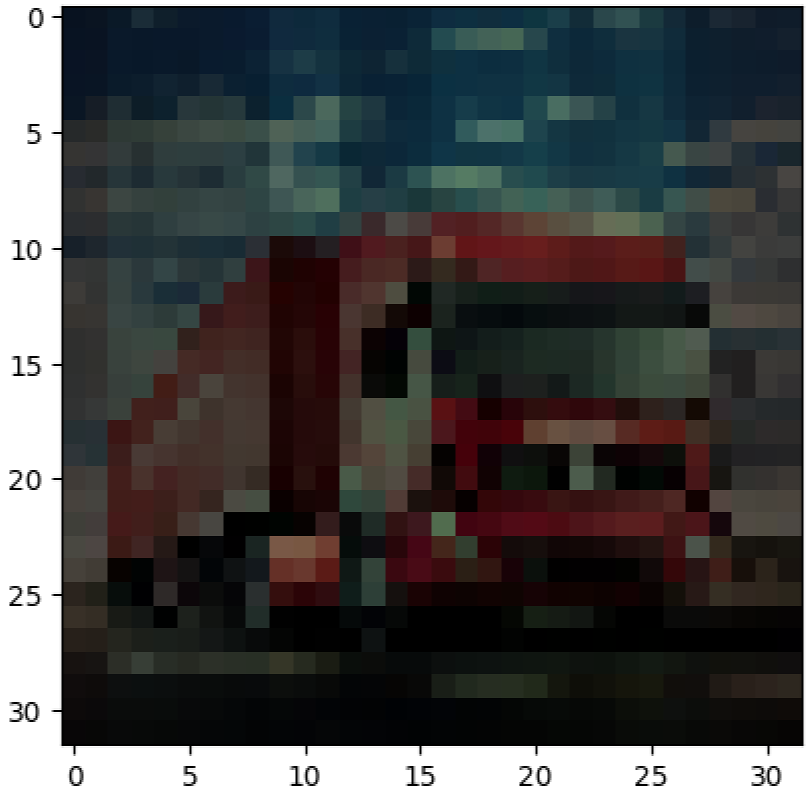

```
1/1 [==============================] - 0s 24ms/step
picture shows:      truck
model prediction:   truck
correct
(32, 32, 3)
```



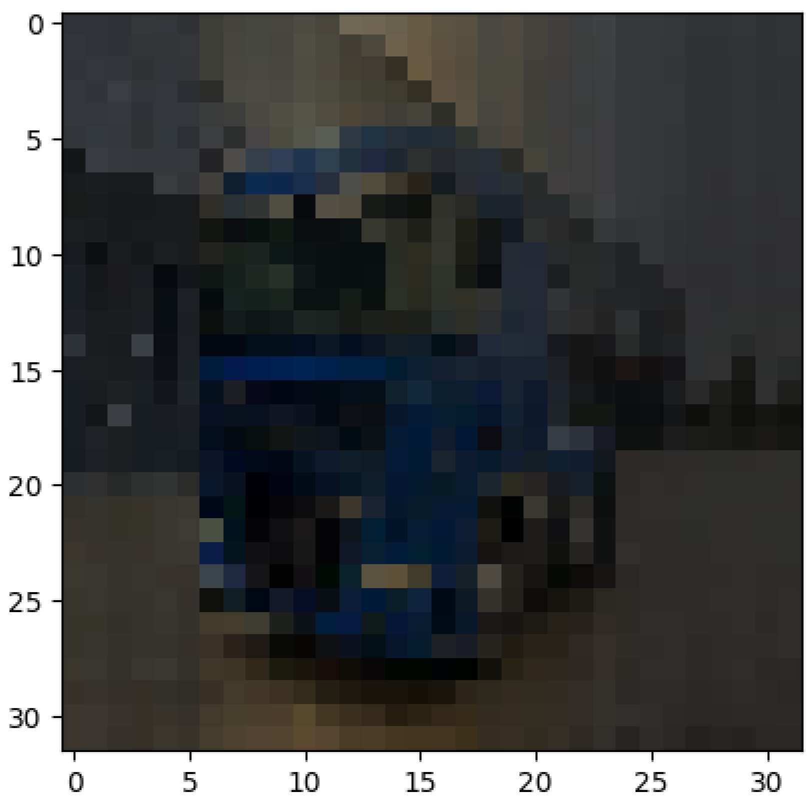
```
1/1 [==============================] - 0s 24ms/step
picture shows:      truck
model prediction:  truck
correct
(32, 32, 3)
```



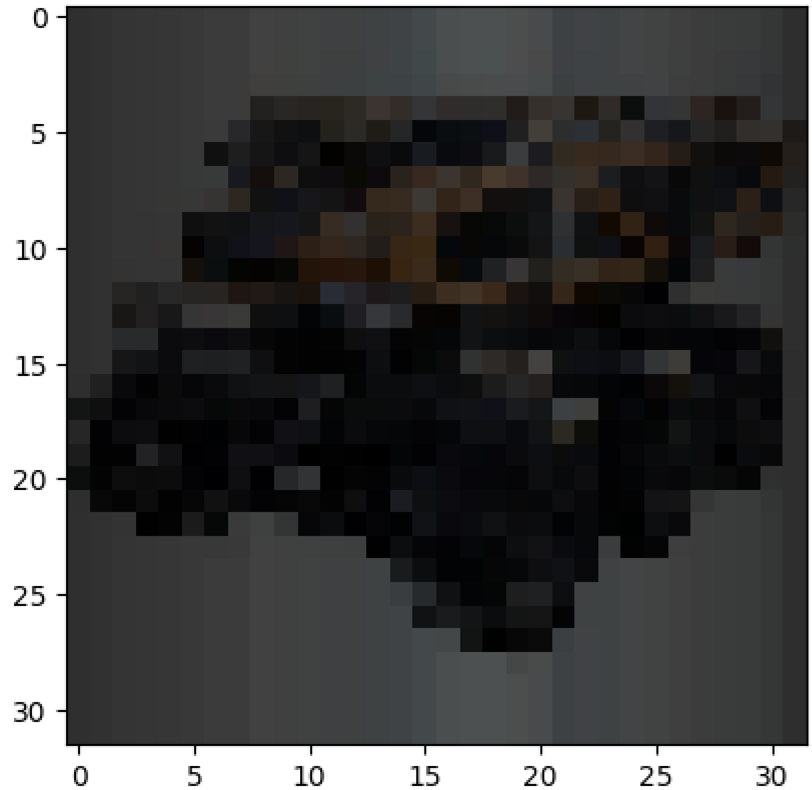
```
1/1 [==============================] - 0s 23ms/step
picture shows:      truck
model prediction:  truck
correct
(32, 32, 3)
```



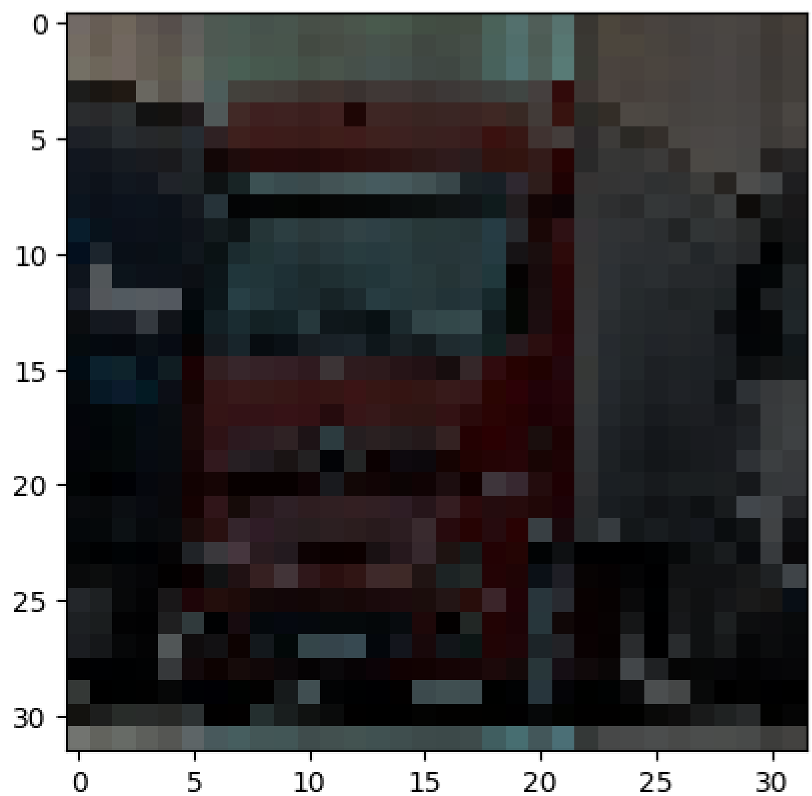
```
1/1 [==============================] - 0s 25ms/step
picture shows:      truck
model prediction:   truck
correct
(32, 32, 3)
```



```
1/1 [==============================] - 0s 24ms/step
picture shows:      truck
model prediction:   truck
correct
(32, 32, 3)
```




```
1/1 [==============================] - 0s 23ms/step
picture shows:      truck
model prediction:  automobile
wrong
(32, 32, 3)
```




```
1/1 [==============================] - 0s 22ms/step
picture shows:      truck
model prediction:  truck
correct
total = 100
acc = 69.0 %
```

In this cell we iterate trough all the pictures and run them trough the model seperately. We also check if the guess was correct and calculated the accuracy of the model.