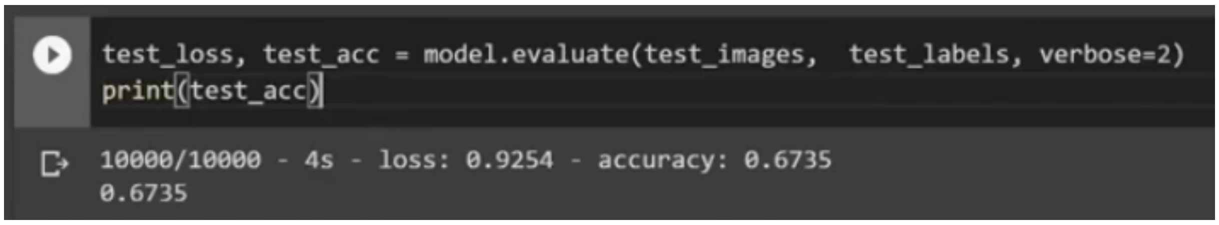
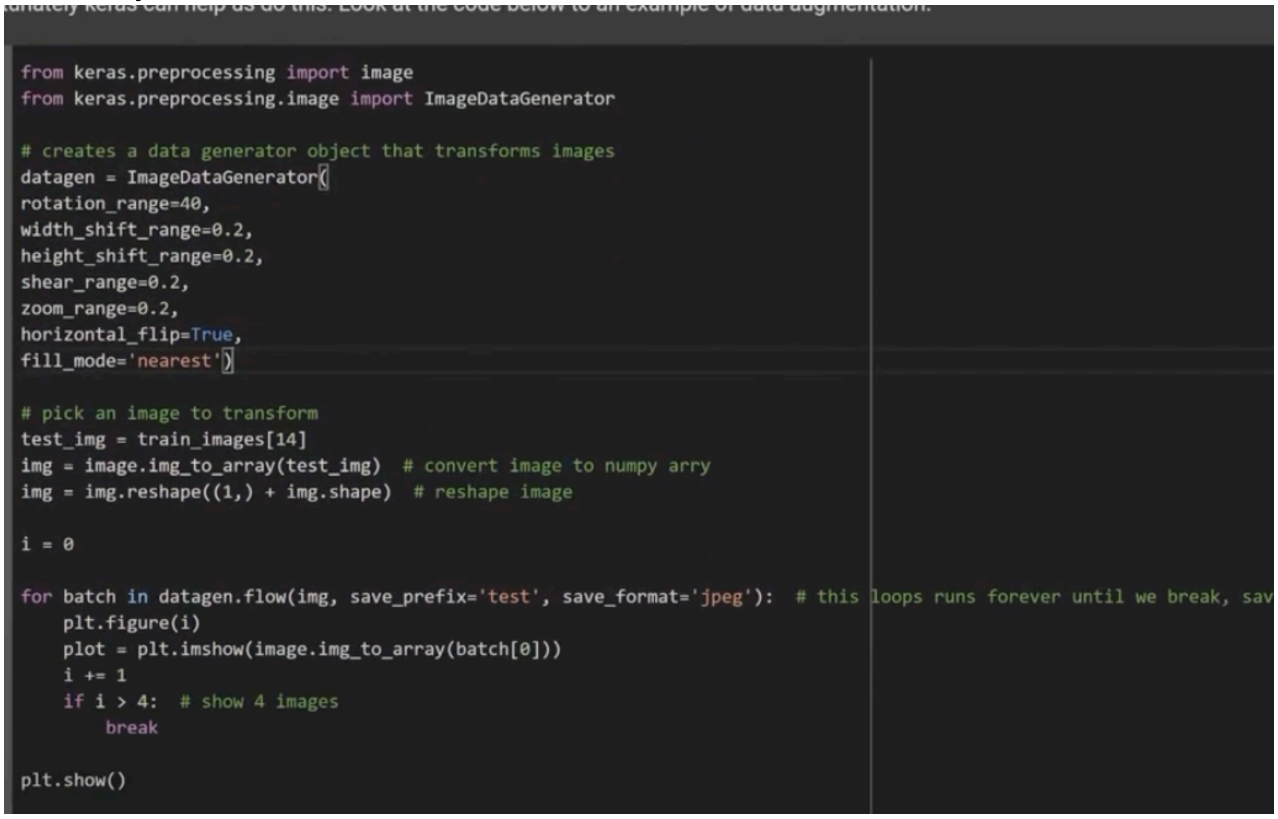
- ***Evaluating the model***
  - -> in other words to see how accurate its predictions are



```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
print(test_acc)

10000/10000 - 4s - loss: 0.9254 - accuracy: 0.6735
0.6735
```

      - -> this is a small dataset of images
      - -> there are specific techniques you can use for smaller datasets
        - -> because the smaller the dataset is the worse the model
- ***Data augmentation***
  - -> using pre-trained models to accomplish what we need to do
  - -> he's made the convolutional neural network in the previous example
  - -> turning the image into many different images and passing it into the model -> compressing it, rotating it - and still passing it into the model
  - **-> an augmentation is a transformation of the image**
  - **-> if you don't have enough images to train the model you can use the same dataset but just transformed versions of the same images**
  - -> he's imported the modules
  - -> then the different transformations to run on the image
    - -> to augment, aka transform the images
  - -> converting the image into an array
  - -> then reshaping it -> the model is deciding the shape
  - -> then iterating through
    - -> in other words taking the image and iterating through the different transformations (augmentations) to run on it
      - -> these augmentations (transformations) are randomised to avoid reducing the accuracy of the model

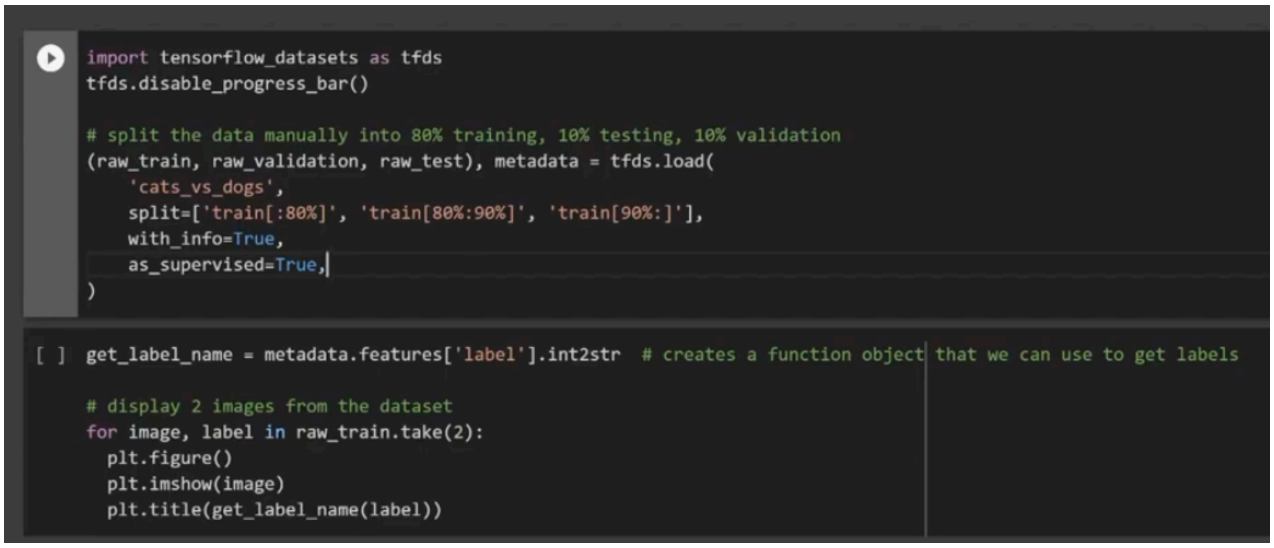


unately Keras can help us do this. Look at the code below to an example of data augmentation.

```
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator

# creates a data generator object that transforms images
datagen = ImageDataGenerator(
rotation_range=40,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
fill_mode='nearest')

# pick an image to transform
test_img = train_images[14]
img = image.img_to_array(test_img)  # convert image to numpy arry
img = img.reshape((1,) + img.shape)  # reshape image

i = 0

for batch in datagen.flow(img, save_prefix='test', save_format='jpeg'):  # this loops runs forever until we break, sav
    plt.figure(i)
    plot = plt.imshow(image.img_to_array(batch[0]))
    i += 1
    if i > 4:  # show 4 images
        break

plt.show()
```

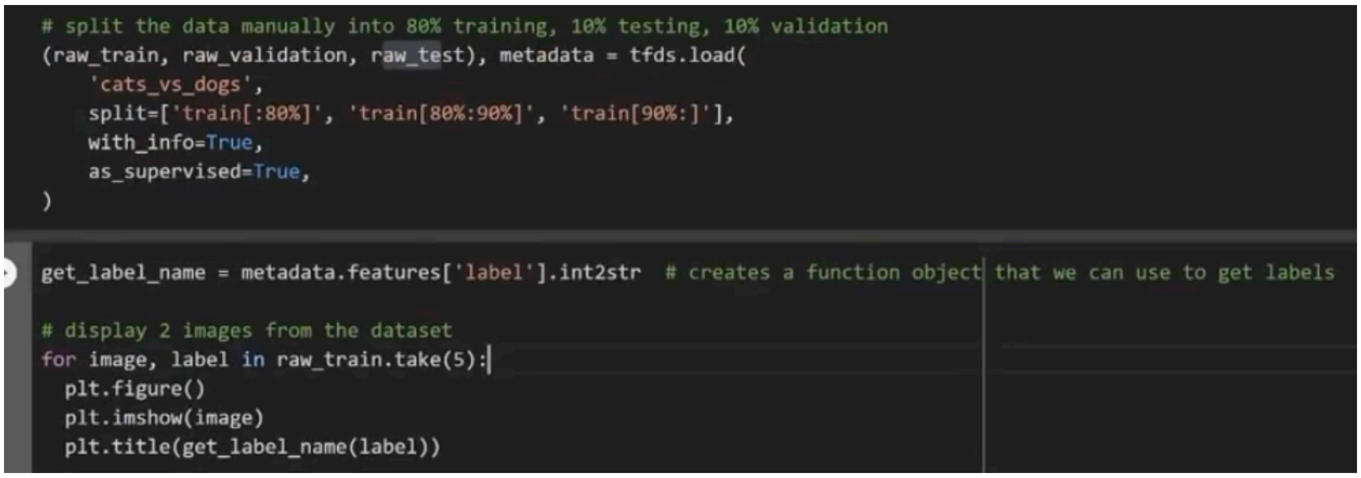
  - -> it's printing out the same image just transformed
    - -> because there isn't enough data in the model -> we're just transforming the data we have and using it as multiple data when training the model

- ***Pretrained models***
  - -> reusing other peoples' models <- big companies have their own models which you can reuse
  - -> they can classify thousands of image types
  - -> these pick up on smaller edges in the model
  - -> you can add your own layers to their model for the problem you want
  - -> fine-tuning part of their model for your specific problem
    - -> you pass in your training data
    - -> e.g just for cats and dogs
    - -> you aren't using their entire model, just the parts of it which are relevant to your problem



```
import tensorflow_datasets as tfds
tfds.disable_progress_bar()

# split the data manually into 80% training, 10% testing, 10% validation
(raw_train, raw_validation, raw_test), metadata = tfds.load(
    'cats_vs_dogs',
    split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
    with_info=True,
    as_supervised=True,
)

[ ] get_label_name = metadata.features['label'].int2str  # creates a function object that we can use to get labels

# display 2 images from the dataset
for image, label in raw_train.take(2):
  plt.figure()
  plt.imshow(image)
  plt.title(get_label_name(label))
```

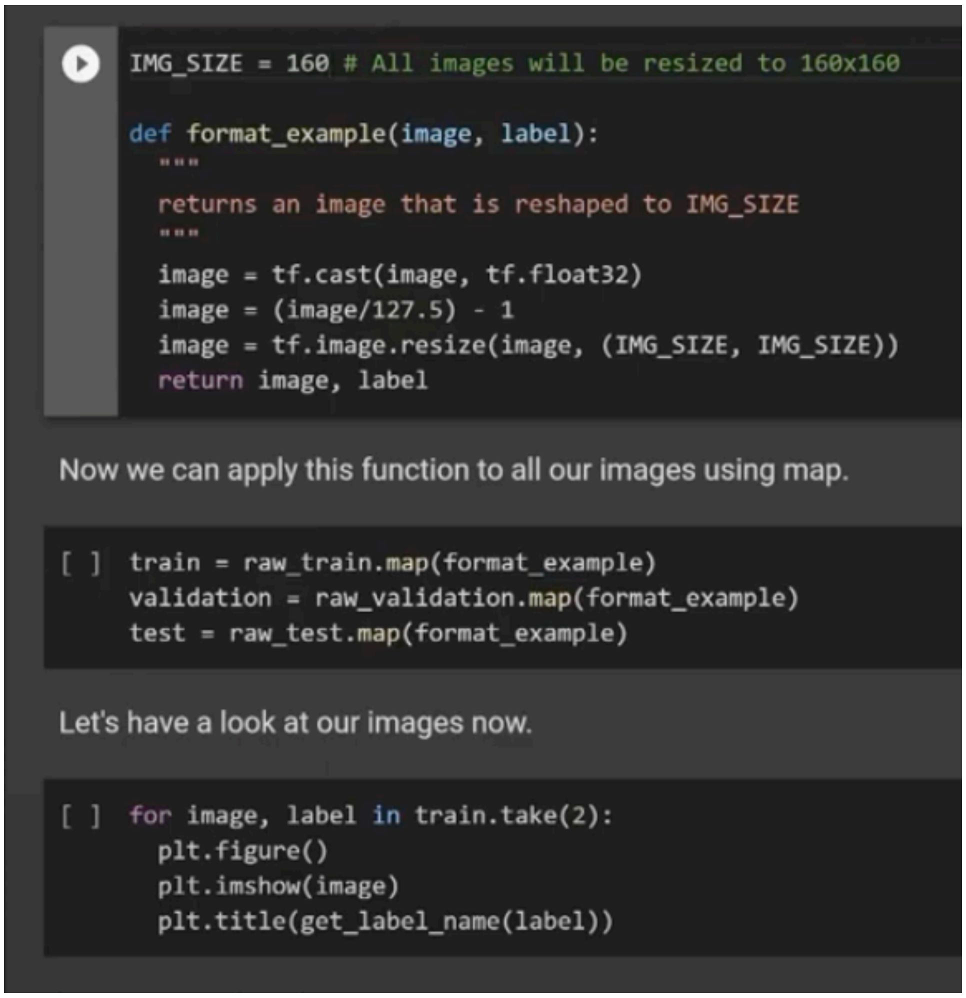
    - -> importing in the dataset
    - -> then loading the data into an numpy array and splitting it into testing, training and validation data
    - -> the images are different dimensions -> scaling them to be the same size



```
# split the data manually into 80% training, 10% testing, 10% validation
(raw_train, raw_validation, raw_test), metadata = tfds.load(
    'cats_vs_dogs',
    split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
    with_info=True,
    as_supervised=True,
)

get_label_name = metadata.features['label'].int2str  # creates a function object that we can use to get labels

# display 2 images from the dataset
for image, label in raw_train.take(5):
  plt.figure()
  plt.imshow(image)
  plt.title(get_label_name(label))
```

      - -> you call the function with an integer and it returns various images
        - -> e.g the number of images you want it to return, and what of

- ***-> to scale those images to be back into the same size***



```
IMG_SIZE = 160 # All images will be resized to 160x160

def format_example(image, label):
  """
  returns an image that is reshaped to IMG_SIZE
  """
  image = tf.cast(image, tf.float32)
  image = (image/127.5) - 1
  image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
  return image, label
```

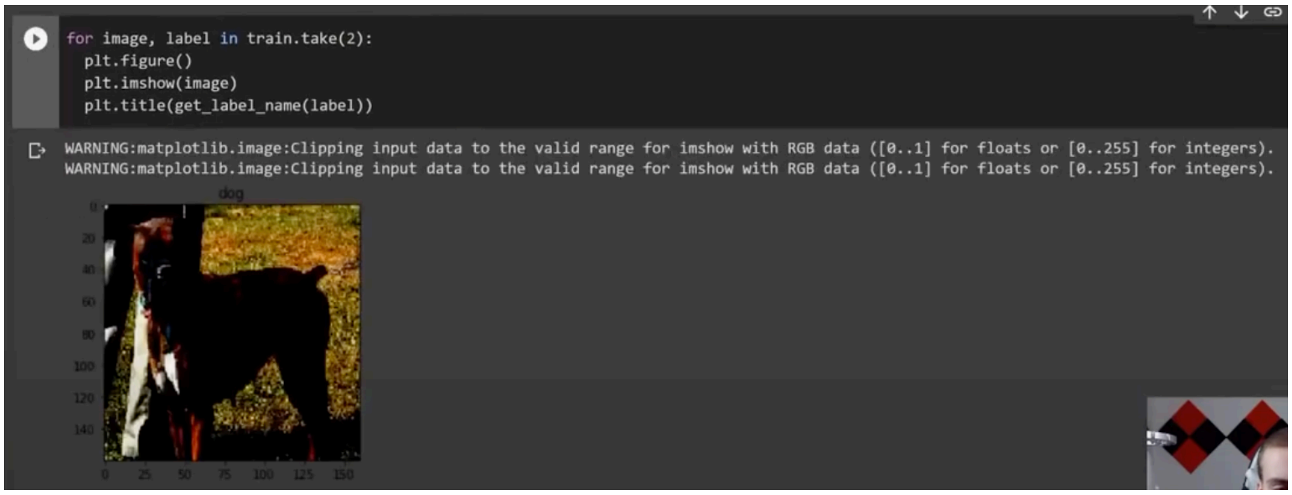
Now we can apply this function to all our images using map.

```
train = raw_train.map(format_example)
validation = raw_validation.map(format_example)
test = raw_test.map(format_example)
```

Let's have a look at our images now.

```
for image, label in train.take(2):
  plt.figure()
  plt.imshow(image)
  plt.title(get_label_name(label))
```

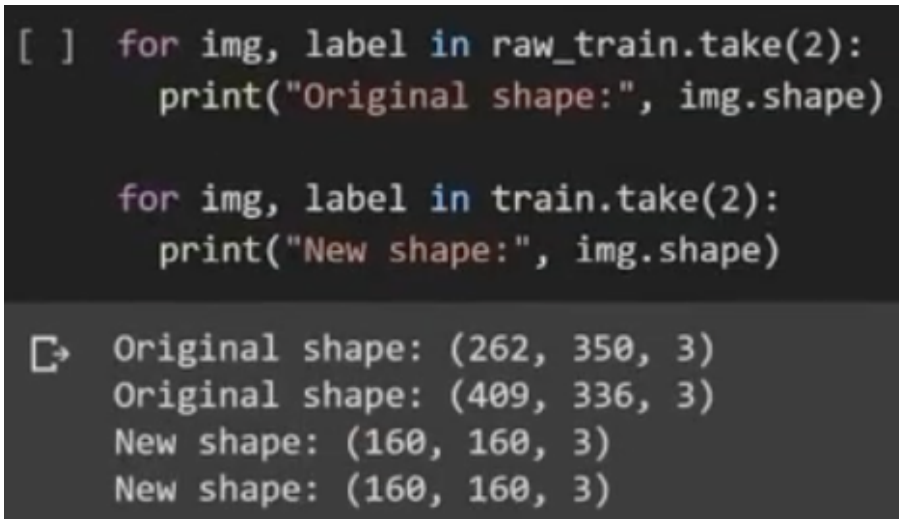
  - ***-> defining a function which reshapes the images in the dataset which we're reusing into the size which we have in our training data***
    - -> we are reusing another model for our data -> this is resizing its images to match the size of the images in our model
    - -> then retuning the reshaped image
    - -> converting every pixel into a value
    - -> <u>cast means convert the pixel values into a float 32 (number value)</u>
    - -> then resizing the image
  - ***-> then applying that function to all of the images in the model we're reusing***
    - -> this is done via a map
    - -> he's printed out the resized images -> and you can see these from the graphs they've been printed on



```
for image, label in train.take(2):
  plt.figure()
  plt.imshow(image)
  plt.title(get_label_name(label))
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

    - -> then he's comparing the shapes of the original images with the new images to check they have been resized
    - -> so, defining a function which resizes the images from the model which we are reusing
    - -> then iterating through the images from that dataset which we are using (not all of the images in the dataset)
    - -> this is the code which checks after running the model that the size of the images in the dataset which we are borrowing are the same as the ones in our dataset which we want to train it with to fine tune the model to our specific context
    - -> we're checking that the images have been correctly resized



```
for img, label in raw_train.take(2):
  print("Original shape:", img.shape)

for img, label in train.take(2):
  print("New shape:", img.shape)
```

```
Original shape: (262, 350, 3)
Original shape: (409, 336, 3)
New shape: (160, 160, 3)
New shape: (160, 160, 3)
```

- ***To increase the accuracy of a convolutional neural networks***
    - -> i.e one which searches for features in the image -> by using filters
    - methods to do those are
        - -> augmenting (transforming) the data which is being used to train it
        - -> using a pre-trained model -> which is reused from a larger company (e.g google)