Computational Intelligence Lab

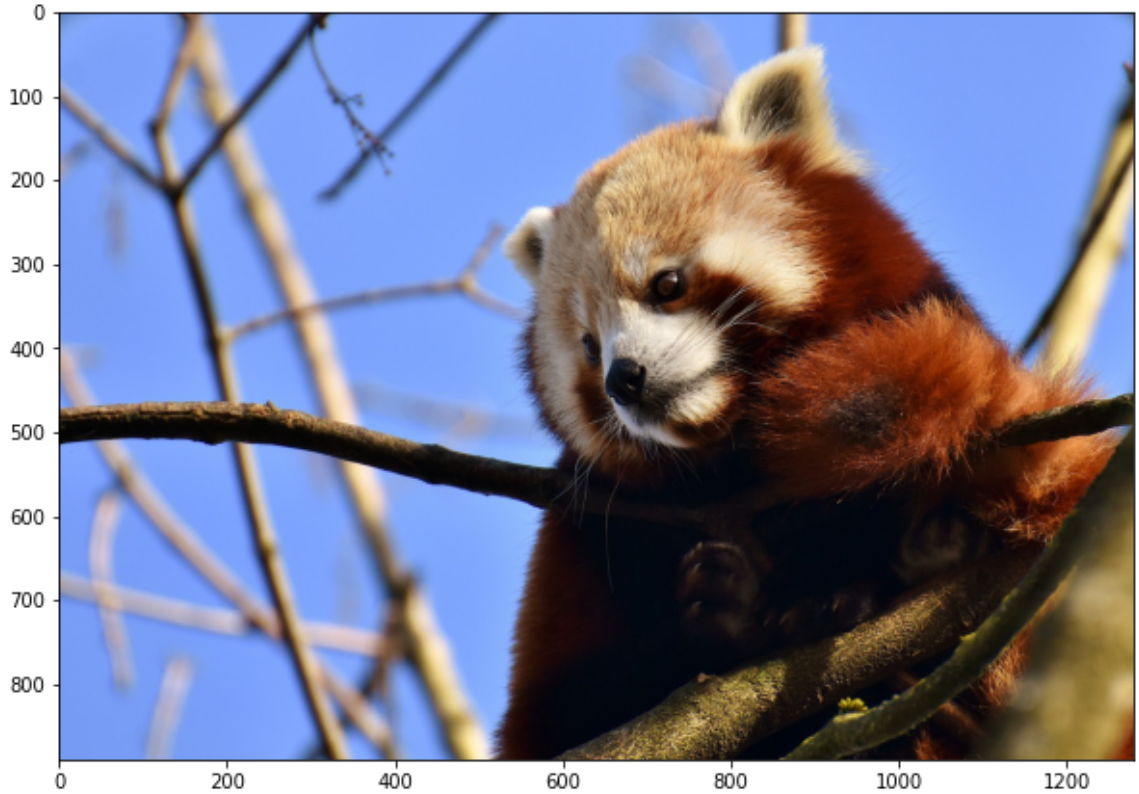
Assignment 4

Karolina Kotlowska IO czw. 9:30

4.1 Introduction - dedicated layers for image augmentation

- Perform various image transformations (rotation, flipping, croping, scaling)
- Parameters specify ranges of transformations
- Can be used as functions (eagerly) - executed on the host
- Can be used as network layers - executed as a computational graph (possibly on GPU)

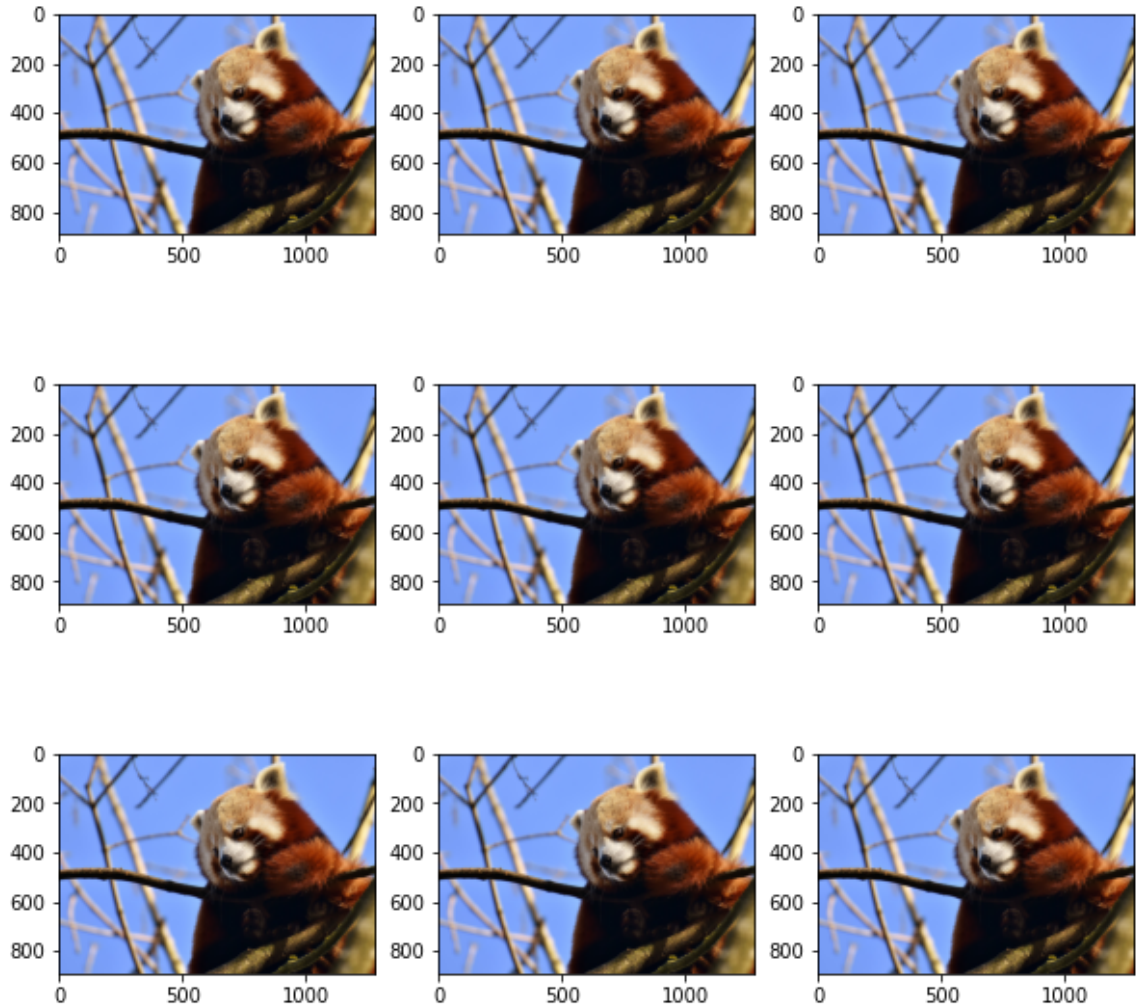
In [7]:
```
from PIL import Image
import requests
from io import BytesIO
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (5, 5)


url='https://cdn.pixabay.com/photo/2018/02/18/00/22/panda-3161290_1280.jp
response = requests.get(url)
# print(response)
img = Image.open(BytesIO(response.content))

plt.rcParams['figure.figsize'] = (10, 10)
_ = plt.imshow(img)
```



In [8]:
```
import numpy as np
import tensorflow as tf
from keras import layers
from keras import models

X = np.array(img)
```

Function displaying a few images after conversion

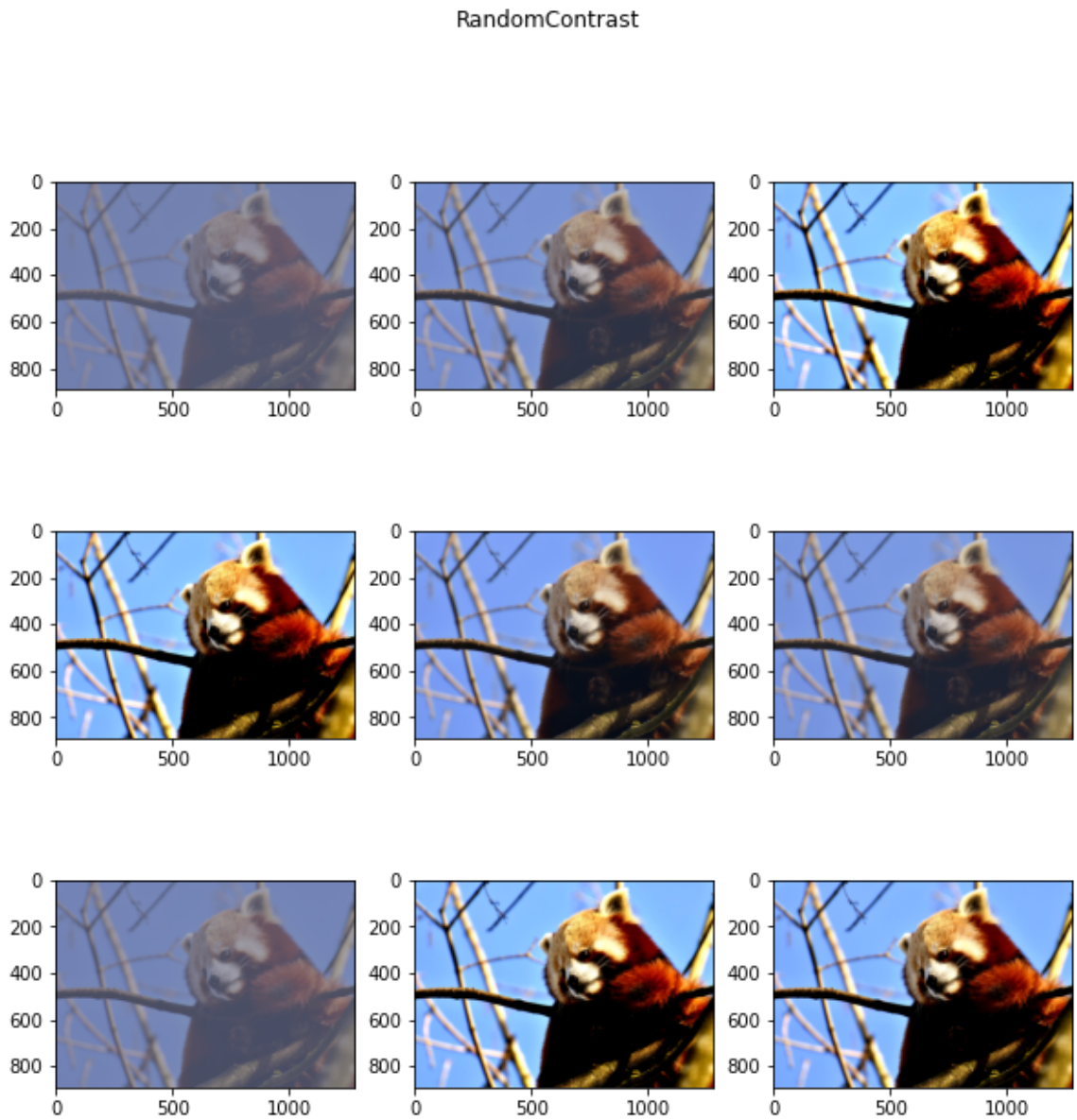
In [9]:
```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (10,10)

def display(X,converter,title=None):

  for i in range(9):
    ax = plt.subplot(330 + 1 + i)
    converted_image  = converter(X)
    plt.imshow(converted_image)
  if title is not None:
    plt.suptitle(title)
  plt.show()

display(X,lambda a:a)
```



RandomContrast

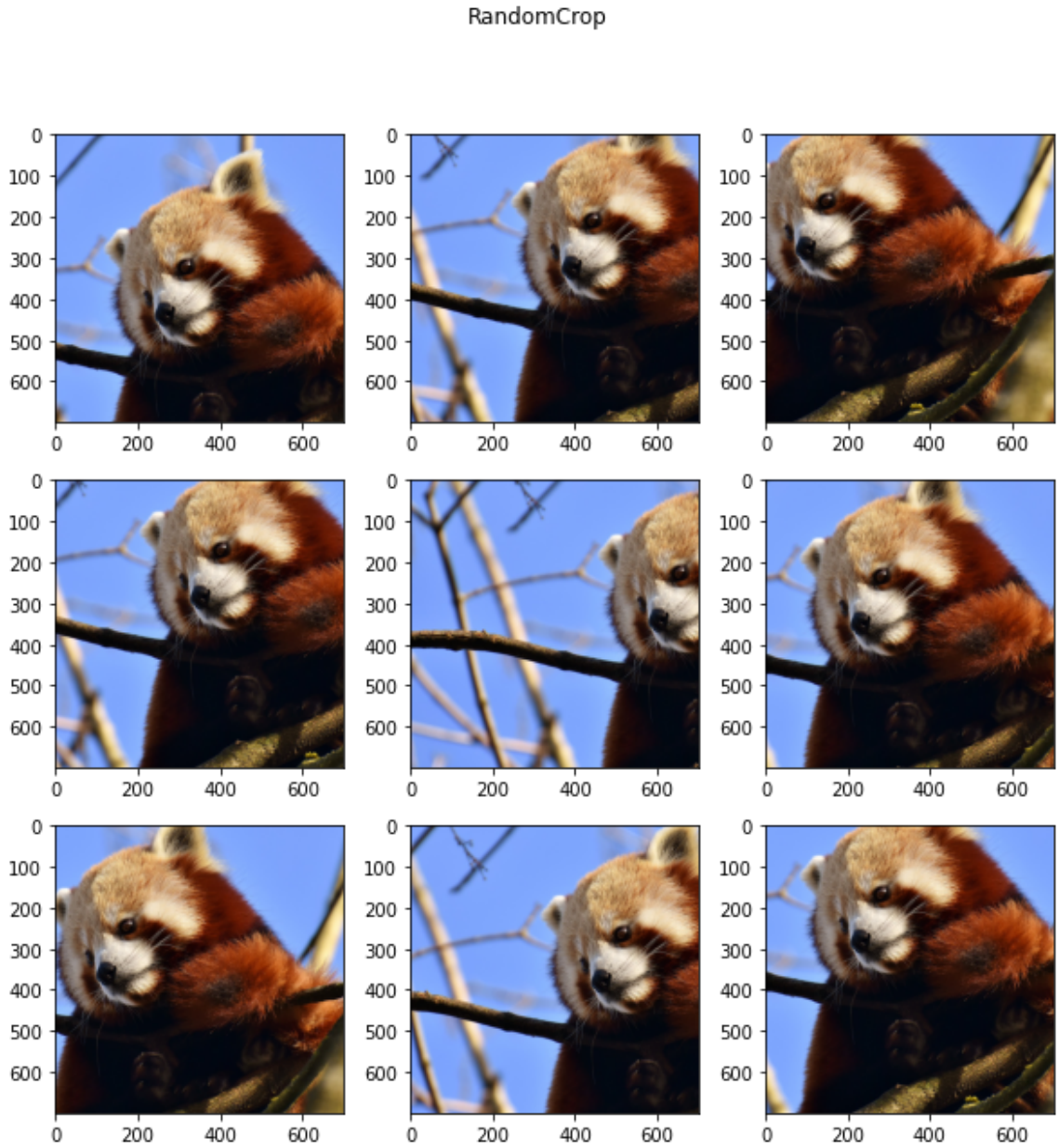
In [10]:
```
converter = models.Sequential([
  layers.RandomContrast(.8,seed=1),
  layers.Rescaling(1./255)
])

display(X,converter,title='RandomContrast')
```

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

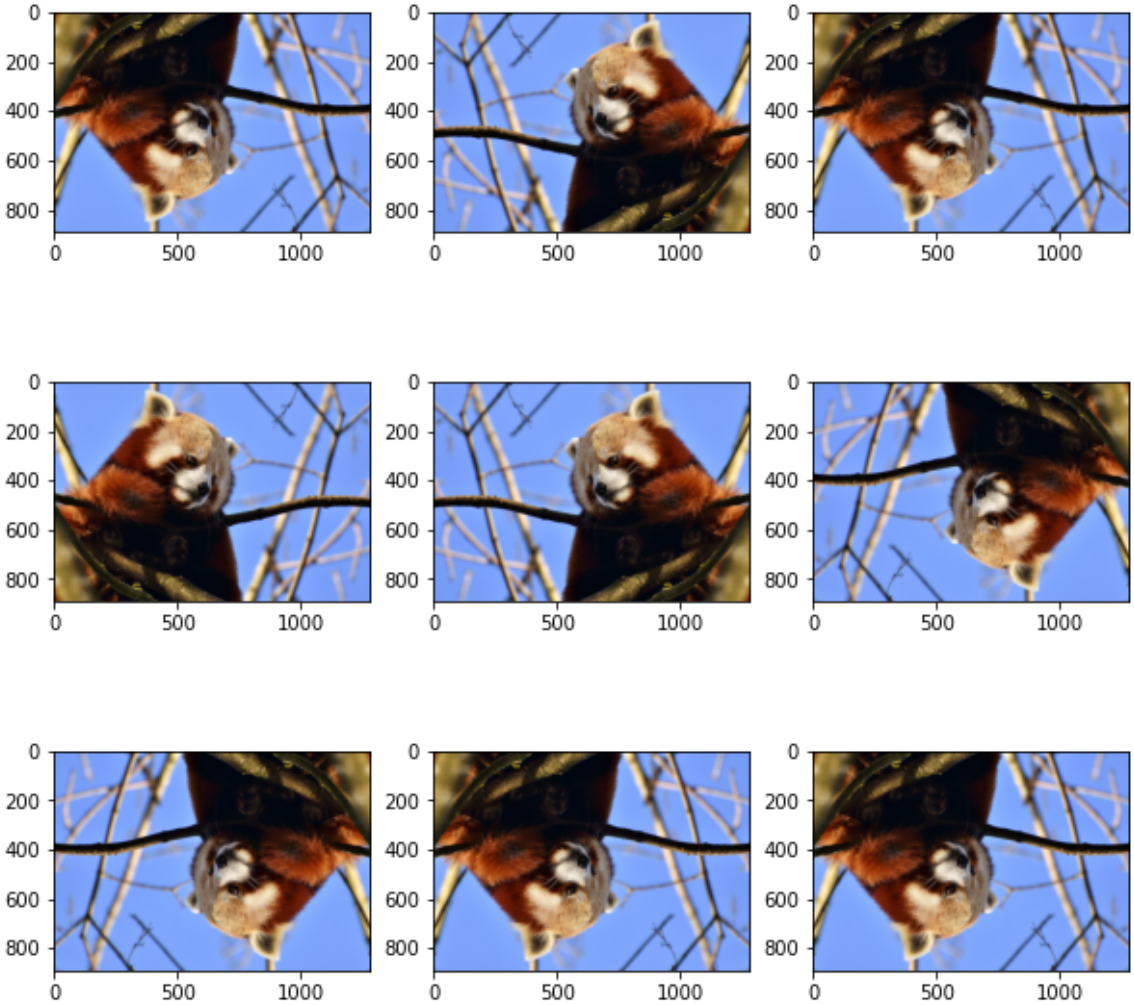


Test more converters

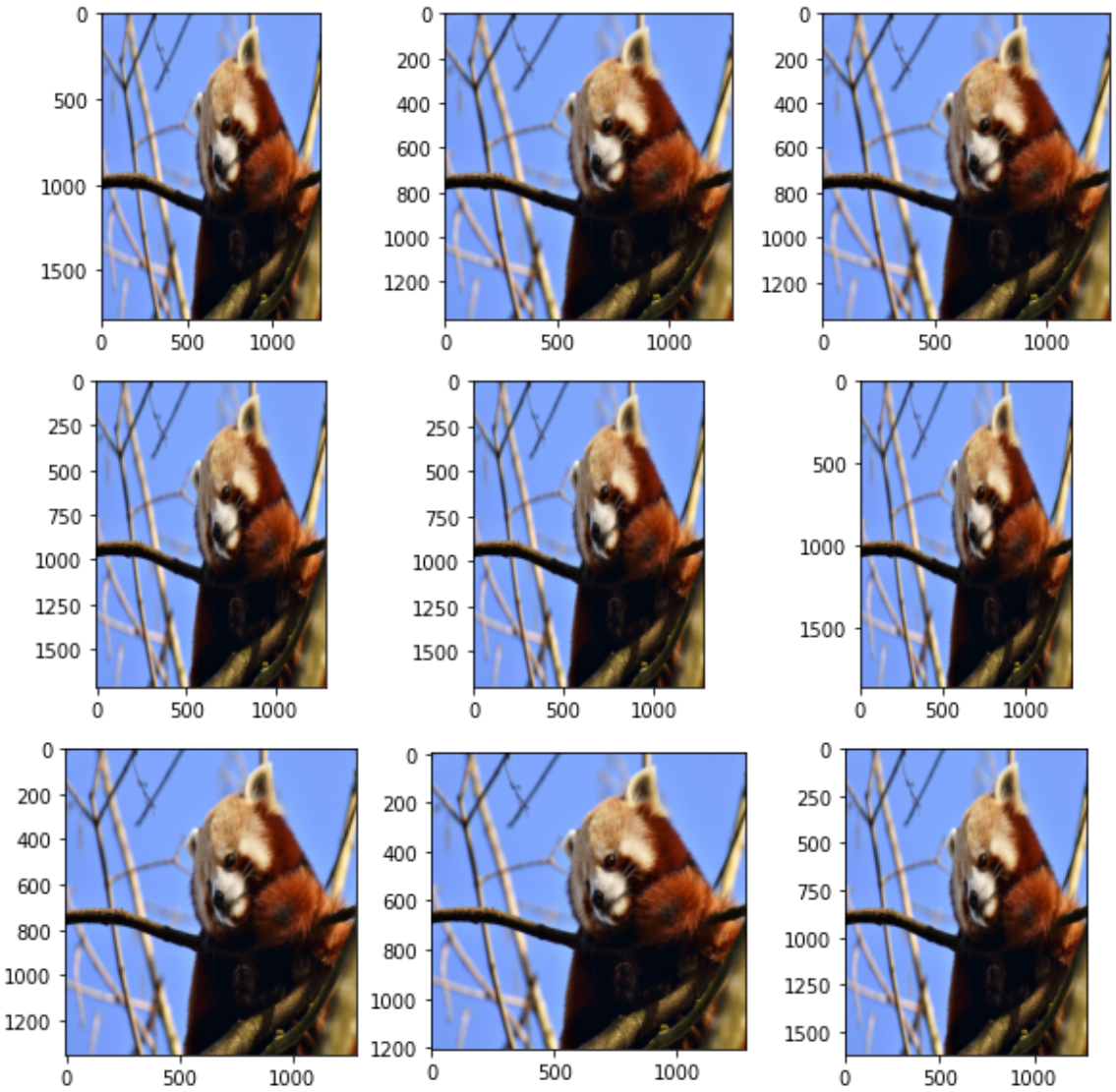
In [11]:
```
preprocessing_layers = [
    layers.RandomCrop(height=700,width=700,seed=1),
    layers.RandomFlip(mode="horizontal_and_vertical", seed=1),
    layers.RandomHeight(factor=(0.3,1.2),seed=1),
    layers.RandomWidth(factor=(0.3,0.8),seed=1),
    layers.RandomRotation((-0.25,0.25),seed=1),
    layers.RandomTranslation(height_factor=(-0.2, 0.3), width_factor=(-0.
    layers.RandomZoom(height_factor=(-0.8,.5),seed=1)

]

for pl in preprocessing_layers:
  print()
  # print(f'=========== {pl.__class__.__name__} ============')
  converter = models.Sequential([
    pl,
    layers.Rescaling(1./255)
  ])
  display(X,converter,title=pl.__class__.__name__)
```


RandomCrop
0
100
200
300
400
500
600
0
200
400
600

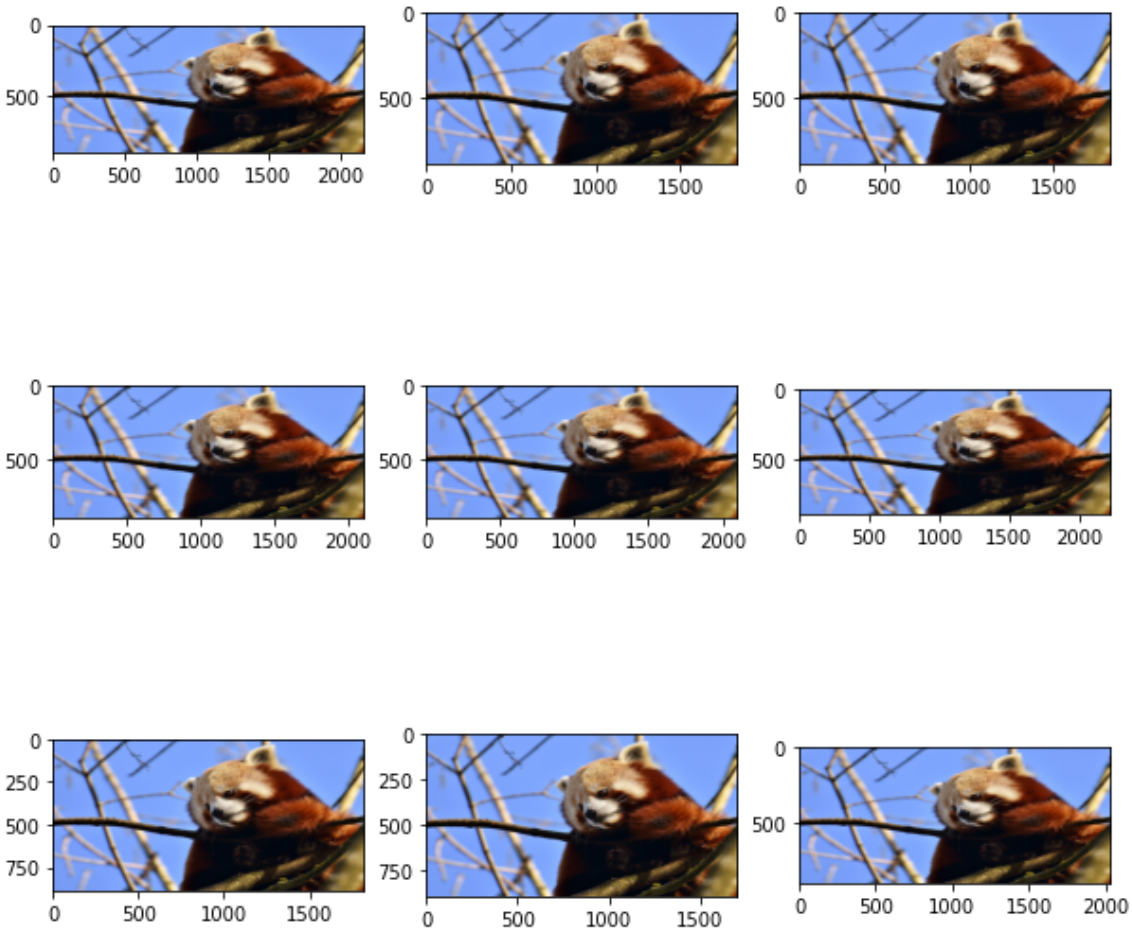
RandomFlip

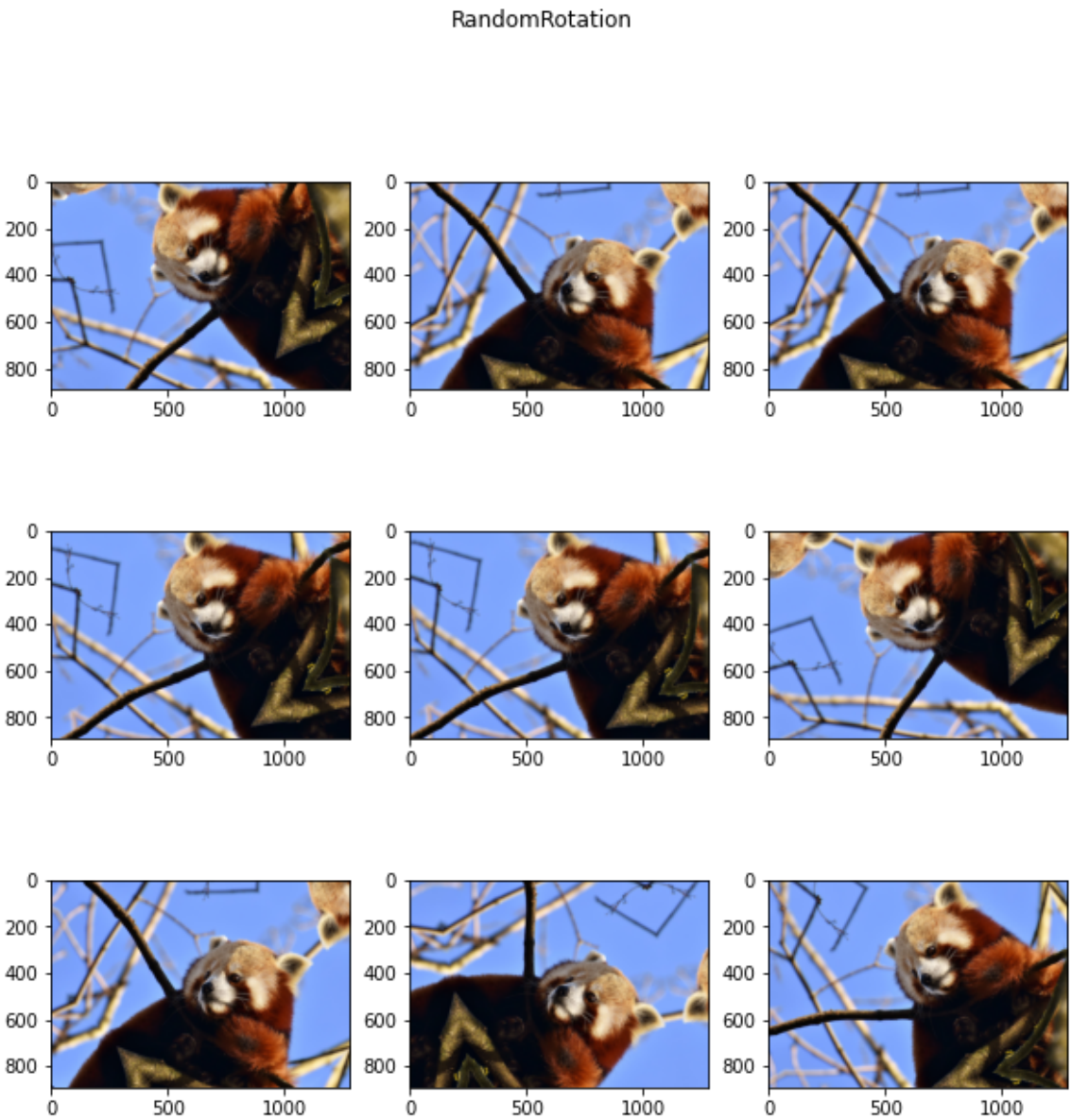


RandomHeight

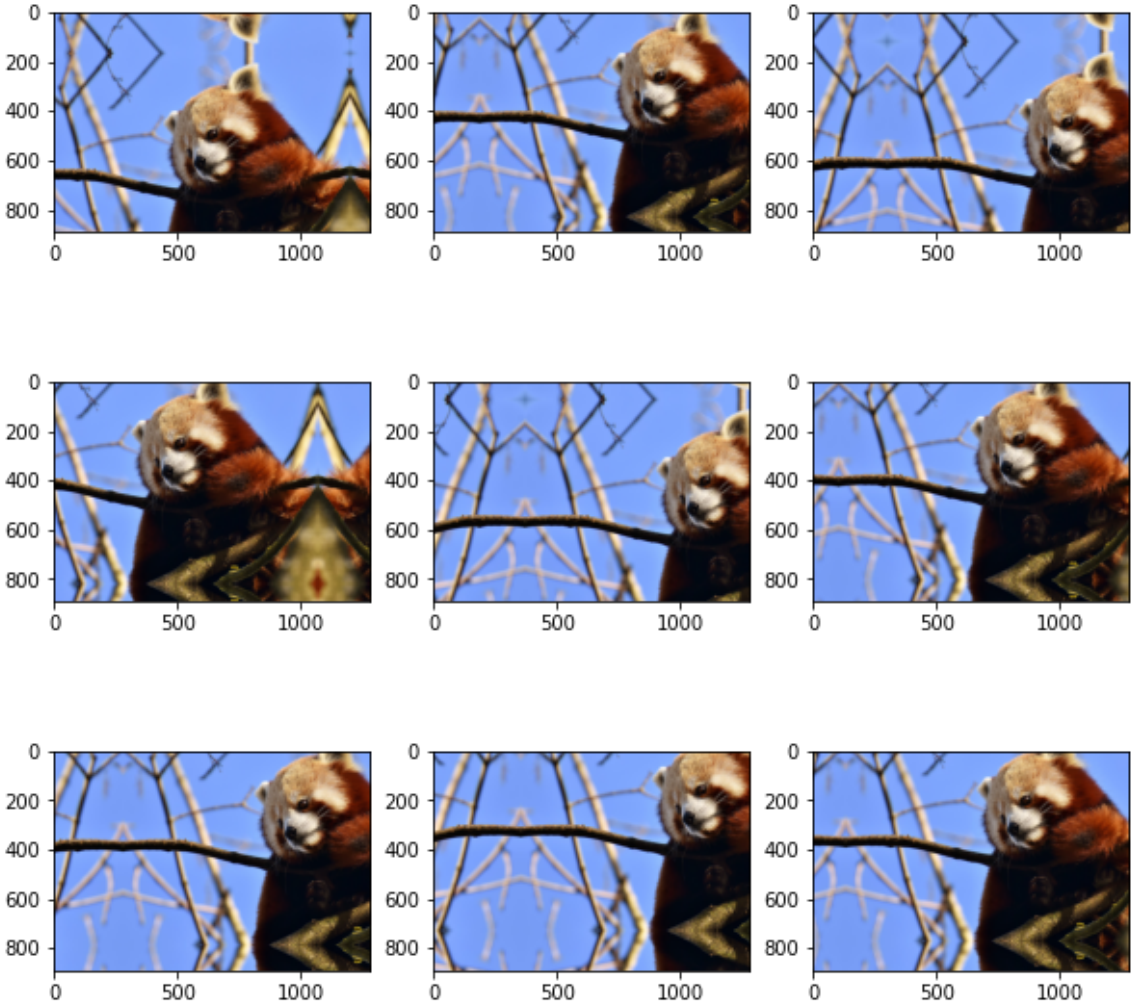


RandomWidth

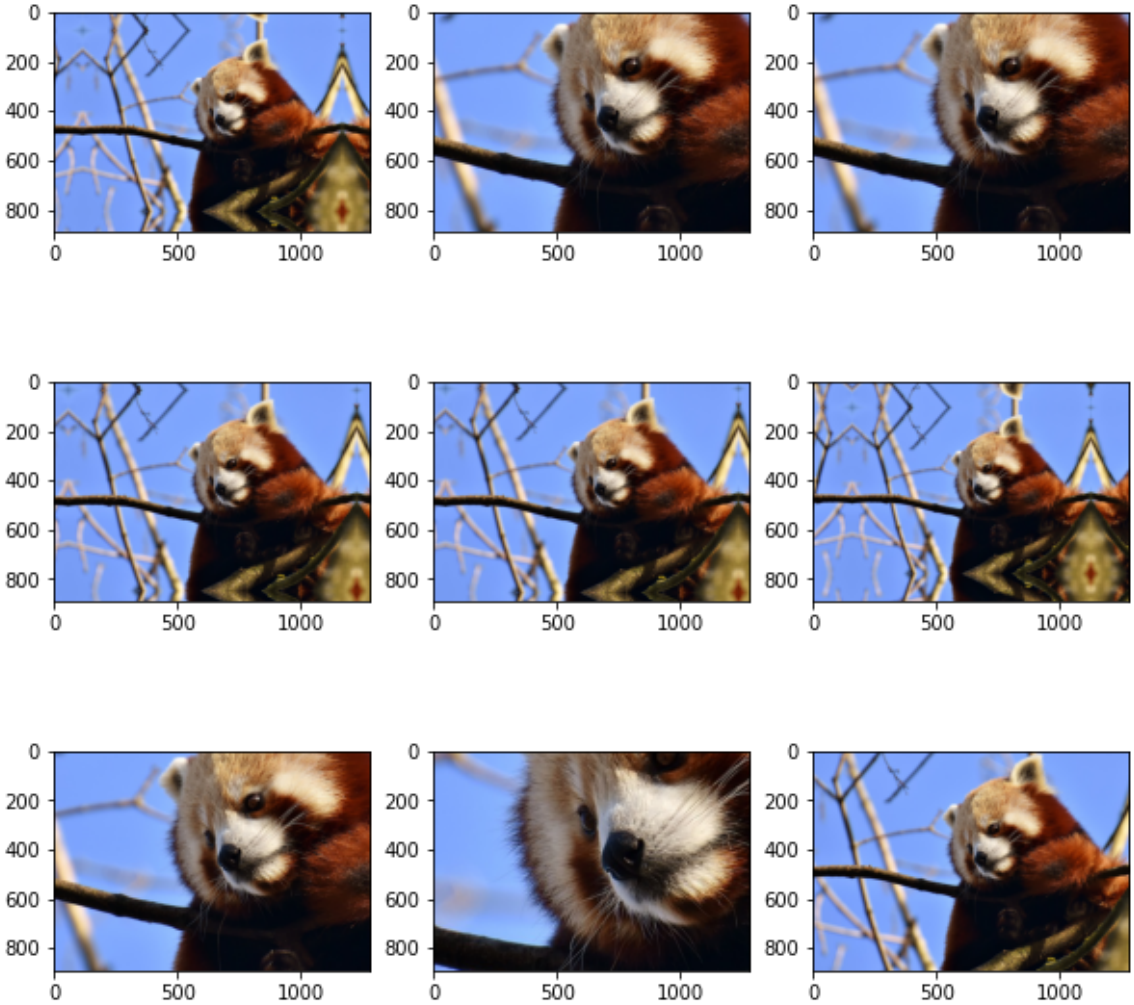



RandomRotation
0
200
400
600
800
0
500
1000

RandomTranslation



RandomZoom



4.2 Horses or humans

We will use horses_or_humans tensorflow dataset.

The TF daset API returns data wrapped as tf.data.Dataset object.

tf.data.Dataset can be considered an equivalent to Java stream. It provides such operations as mapping, filtering, limiting, shuffling and skipping data. It is also reponsible for formation of batches.

Dataset is intended to execute on CPU in parallel with training on GPU. While the network is trained using data in a current batch, a next batch is fetched and prepared for delivery.

We must downgrade Tensorflow !!!

In [12]:
```
# !pip3 uninstall tensorflow
# !pip3 install tensorflow-gpu==2.8.3
```

In [13]:
```
import tensorflow_datasets as tfds
import tensorflow as tf

ds_train = tfds.load('horses_or_humans', split='train', as_supervised=Tru
ds_test = tfds.load('horses_or_humans', split='test', as_supervised=True,
```

In [14]:
```
# How many elements?
print(f'Number of images in train set: {ds_train.cardinality()}')
print(f'Number of images in test set: {ds_test.cardinality()}')
```

```
Number of images in train set: 1027
Number of images in test set: 256
```

In [15]:
```
# How many classes

labels = [label.numpy() for image, label in ds_train]
num_classes = max(labels)+1
print(f'#classes = {num_classes}') #humans and horses
```

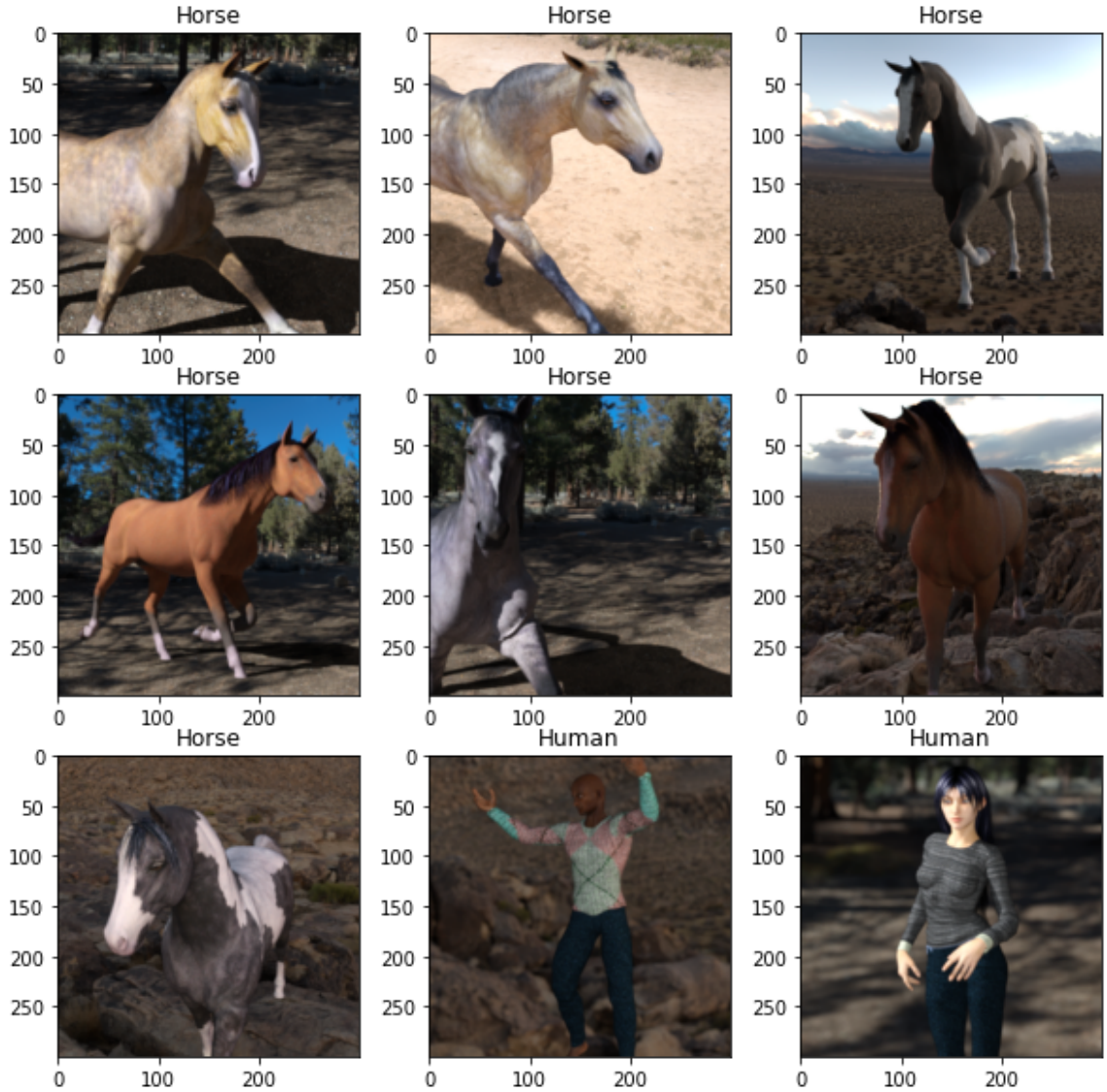
```
#classes = 2
```

In [16]:
```
# image shapes?

images = [image.numpy() for image, label in ds_train.take(10)] # first 10
for img in images:
  print(f'shape={img.shape}')
```

```
shape=(300, 300, 3)
shape=(300, 300, 3)
shape=(300, 300, 3)
shape=(300, 300, 3)
shape=(300, 300, 3)
shape=(300, 300, 3)
shape=(300, 300, 3)
shape=(300, 300, 3)
shape=(300, 300, 3)
shape=(300, 300, 3)
```

Display a few images

In [17]:
```
#display a few images...
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (10, 10)

ds2 = ds_train.take(9) # take - equivalent of limit
it = ds2.as_numpy_iterator()

labels=['Horse','Human']
input_size=100
for i in range(9):
  ax = plt.subplot(330 + 1 + i)
  image, label = next(it)
  ax.set_title(labels[label])
  plt.imshow(image)
plt.show()
```



Apply sample transformations

In [18]:
```
rotate = layers.RandomRotation((-0.25,0.25),seed=123)
rescale = layers.Rescaling(1./255)
ds2 = ds_train.take(9).map(lambda x,y: (rescale(rotate(x)),y)) #<<<<< rot
it = ds2.as_numpy_iterator()


labels=['Horse','Human']
input_size=100
for i in range(9):
  ax = plt.subplot(330 + 1 + i)
  image, label = next(it)
  ax.set_title(labels[label])
  plt.imshow(image)
plt.show()
```

4.2.1 Classification (no augmentation)

TODO 4.2.1 Propose your own model consisting of a few Conv2D, MaxPooling2D and (optionaly) dropout layers. Then Flatten, Dense. As this is a binary classification problem - you may use one (sigmoid) or two (softmax) neurons in the output layer. Please observe: the shape of input layer must match the shapes of images.

In [19]:
```
from keras import layers
from keras import models
from keras import optimizers

image_width = 300
image_height = 300

def get_classifier_model():
  classifier_model = models.Sequential([
    layers.Conv2D(16, (3,3), activation='relu', input_shape=(image_height
    layers.MaxPooling2D(2, 2),
    layers.Dropout(0.1),

    layers.Conv2D(32, (3,3), activation ='relu'),
    layers.MaxPooling2D(2,2),
```

```
    layers.Dropout(0.2),

    layers.Conv2D(64, (3,3), activation ='relu'),
    layers.MaxPooling2D(2,2),
    layers.Dropout(0.3),


    layers.Flatten(),

    layers.Dense(512, activation = 'relu'),
    layers.Dense(2, activation = 'sigmoid')
    ])
  return classifier_model
```

Configuration of Dataset streams

In [20]:
```
def prepare_dataset(ds,batch_size=64,resize=None, augment=None):
  if resize is not None:
    ds = ds.map(lambda x, y: (resize(x), y),
                num_parallel_calls=tf.data.AUTOTUNE)

  ds = ds.shuffle(1000)
  ds = ds.batch(batch_size)

  if augment is not None:
    ds = ds.map(lambda x, y: (augment(x), y),
                num_parallel_calls=tf.data.AUTOTUNE)

  return ds.prefetch(buffer_size=tf.data.AUTOTUNE)

ds_train_preprocessed = prepare_dataset(ds_train)
ds_test_preprocessed = prepare_dataset(ds_test)
```

Training

In [21]:
```
classifier_model = get_classifier_model()

classifier_model.compile(loss='sparse_categorical_crossentropy',
              optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),
              metrics=['accuracy'])
classifier_model.summary()

epochs = 10
hist = classifier_model.fit(ds_train_preprocessed,epochs=epochs)
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 298, 298, 16)      448

 max_pooling2d (MaxPooling2D  (None, 149, 149, 16)     0
 )

 dropout (Dropout)           (None, 149, 149, 16)      0

 conv2d_1 (Conv2D)           (None, 147, 147, 32)      4640

 max_pooling2d_1 (MaxPooling  (None, 73, 73, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 73, 73, 32)        0

 conv2d_2 (Conv2D)           (None, 71, 71, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 35, 35, 64)       0
 2D)

 dropout_2 (Dropout)         (None, 35, 35, 64)        0

 flatten (Flatten)           (None, 78400)             0

 dense (Dense)               (None, 512)               40141312

 dense_1 (Dense)             (None, 2)                 1026

=================================================================
Total params: 40,165,922
Trainable params: 40,165,922
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
17/17 [==============================] - 14s 116ms/step - loss: 483.5246
- accuracy: 0.5307
Epoch 2/10
17/17 [==============================] - 2s 106ms/step - loss: 19.9836 -
accuracy: 0.5404
Epoch 3/10
17/17 [==============================] - 2s 105ms/step - loss: 17.7230 -
accuracy: 0.5307
Epoch 4/10
17/17 [==============================] - 2s 104ms/step - loss: 13.3391 -
accuracy: 0.5794
Epoch 5/10
17/17 [==============================] - 2s 107ms/step - loss: 10.5919 -
accuracy: 0.5940
Epoch 6/10
17/17 [==============================] - 2s 108ms/step - loss: 7.1787 -
accuracy: 0.6407
Epoch 7/10
17/17 [==============================] - 2s 105ms/step - loss: 5.6958 -
accuracy: 0.6660
Epoch 8/10
17/17 [==============================] - 2s 106ms/step - loss: 4.6420 -
accuracy: 0.6767
```

```
Epoch 9/10
17/17 [==============================] - 2s 106ms/step - loss: 1.6587 -
accuracy: 0.8004
Epoch 10/10
17/17 [==============================] - 2s 107ms/step - loss: 2.7734 -
accuracy: 0.7575
```

Evaluate results

In [22]:
```
loss, acc = classifier_model.evaluate(ds_test_preprocessed)
print("Accuracy", acc)
```

```
4/4 [==============================] - 1s 35ms/step - loss: 3.0577 - acc
uracy: 0.5117
Accuracy 0.51171875
```

4.2.2. Augmentation on host

In this setup images will be transformed during operations performed by tf.data.Dataset. Batch data can be prepared in parallel by multiple CPU threads.

TODO 4.2.2 Design your own transformations sequence consisting of at least two steps (apart from resizing and rescaling). Avoid shots that are not likely to occur. Rather introduce small modifications...

In [69]:
```
# image_width = 300
# image_height = 300

transform = models.Sequential([
  layers.RandomCrop(height=700,width=700,seed=1),
  #200, 200 seed 1
  layers.RandomRotation((-0.25,0.25),seed=1),
  layers.RandomZoom(height_factor=(-0.8,.5),seed=1),
  layers.RandomFlip(mode="horizontal_and_vertical", seed=1),
  layers.RandomHeight(factor=(0.4,1),seed=1),
  layers.RandomZoom (-0.8, 0.5, seed=1),
  layers.RandomTranslation(height_factor=(-0.2, 0.2), width_factor=(-0.2,

  layers.Resizing(image_height, image_width),
  layers.Rescaling(1./255),
])

resize = models.Sequential([
  layers.Resizing(image_width, image_height),
  layers.Rescaling(1./255)
])
```

Augmentation is applied to ds_train. Test data are not transformed.

In [24]:
```
ds_train_preprocessed = prepare_dataset(ds_train,augment = transform )
ds_test_preprocessed = prepare_dataset(ds_test)
```

TODO 4.2.3 Compile, fit during 20 epochs and evaluate results

In [25]:
```
classifier_model = get_classifier_model()

classifier_model.compile(loss='sparse_categorical_crossentropy',
              optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),
              metrics=['accuracy'])

classifier_model.summary()

hist = classifier_model.fit(ds_train_preprocessed,epochs=epochs)
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 298, 298, 16)      448

 max_pooling2d_3 (MaxPooling  (None, 149, 149, 16)     0
 2D)

 dropout_3 (Dropout)         (None, 149, 149, 16)      0

 conv2d_4 (Conv2D)           (None, 147, 147, 32)      4640

 max_pooling2d_4 (MaxPooling  (None, 73, 73, 32)       0
 2D)

 dropout_4 (Dropout)         (None, 73, 73, 32)        0

 conv2d_5 (Conv2D)           (None, 71, 71, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 35, 35, 64)       0
 2D)

 dropout_5 (Dropout)         (None, 35, 35, 64)        0

 flatten_1 (Flatten)         (None, 78400)             0

 dense_2 (Dense)             (None, 512)               40141312

 dense_3 (Dense)             (None, 2)                 1026

=================================================================
Total params: 40,165,922
Trainable params: 40,165,922
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
17/17 [==============================] - 10s 451ms/step - loss: 0.6812 -
accuracy: 0.5735
Epoch 2/10
17/17 [==============================] - 9s 488ms/step - loss: 0.6468 -
accuracy: 0.6261
Epoch 3/10
17/17 [==============================] - 7s 380ms/step - loss: 0.6170 -
accuracy: 0.6582
Epoch 4/10
17/17 [==============================] - 9s 496ms/step - loss: 0.5823 -
accuracy: 0.6962
Epoch 5/10
17/17 [==============================] - 9s 459ms/step - loss: 0.5619 -
accuracy: 0.7283
Epoch 6/10
17/17 [==============================] - 8s 408ms/step - loss: 0.5203 -
accuracy: 0.7585
Epoch 7/10
17/17 [==============================] - 9s 494ms/step - loss: 0.4976 -
accuracy: 0.7614
Epoch 8/10
17/17 [==============================] - 7s 377ms/step - loss: 0.4600 -
accuracy: 0.7936
```

```
Epoch 9/10
17/17 [==============================] - 7s 350ms/step - loss: 0.4400 -
accuracy: 0.7916
Epoch 10/10
17/17 [==============================] - 9s 487ms/step - loss: 0.4136 -
accuracy: 0.7984
```

In [26]:
```
loss, acc = classifier_model.evaluate(ds_test_preprocessed)
print("Accuracy", acc)
```

```
4/4 [==============================] - 0s 35ms/step - loss: 2410.0745 -
accuracy: 0.8438
Accuracy 0.84375
```

4.2.3 Augmentation on GPU

In this case transformation layers are part of the model. We compose the model from two submodels: transform and classifier_model.

Data augmentation step is not included into Dataset stream.

In [27]:
```
classifier_model = get_classifier_model()

full_model = models.Sequential([
  transform,
  classifier_model
])

ds_train_preprocessed = prepare_dataset(ds_train)
ds_test_preprocessed = prepare_dataset(ds_test)
```

TODO 4.2.4 Compile full_model, train during 20 epochs? Compute accuracy on the tets set.

In [29]:
```
full_model.compile(loss='sparse_categorical_crossentropy',
              optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),
              metrics=['accuracy'])

epochs = 20
hist = full_model.fit(ds_train_preprocessed, epochs=epochs)
```

```
Epoch 1/20
17/17 [==============================] - 7s 298ms/step - loss: 0.6921 -
accuracy: 0.5200
Epoch 2/20
17/17 [==============================] - 5s 301ms/step - loss: 0.6922 -
accuracy: 0.5200
Epoch 3/20
17/17 [==============================] - 5s 301ms/step - loss: 0.6895 -
accuracy: 0.5122
Epoch 4/20
17/17 [==============================] - 5s 313ms/step - loss: 0.6893 -
accuracy: 0.5278
Epoch 5/20
17/17 [==============================] - 5s 298ms/step - loss: 0.6854 -
accuracy: 0.5540
Epoch 6/20
17/17 [==============================] - 5s 282ms/step - loss: 0.6841 -
accuracy: 0.5550
Epoch 7/20
17/17 [==============================] - 5s 297ms/step - loss: 0.6884 -
accuracy: 0.5443
Epoch 8/20
17/17 [==============================] - 5s 310ms/step - loss: 0.6834 -
accuracy: 0.5501
Epoch 9/20
17/17 [==============================] - 5s 278ms/step - loss: 0.6725 -
accuracy: 0.5852
Epoch 10/20
17/17 [==============================] - 5s 300ms/step - loss: 0.6865 -
accuracy: 0.5638
Epoch 11/20
17/17 [==============================] - 5s 308ms/step - loss: 0.6729 -
accuracy: 0.5988
Epoch 12/20
17/17 [==============================] - 5s 298ms/step - loss: 0.6723 -
accuracy: 0.5881
Epoch 13/20
17/17 [==============================] - 5s 295ms/step - loss: 0.6730 -
accuracy: 0.5813
Epoch 14/20
17/17 [==============================] - 5s 292ms/step - loss: 0.6705 -
accuracy: 0.6037
Epoch 15/20
17/17 [==============================] - 5s 306ms/step - loss: 0.6713 -
accuracy: 0.5930
Epoch 16/20
17/17 [==============================] - 5s 280ms/step - loss: 0.6709 -
accuracy: 0.5852
Epoch 17/20
17/17 [==============================] - 5s 305ms/step - loss: 0.6778 -
accuracy: 0.5716
Epoch 18/20
17/17 [==============================] - 5s 289ms/step - loss: 0.6656 -
accuracy: 0.6066
Epoch 19/20
17/17 [==============================] - 5s 297ms/step - loss: 0.6814 -
accuracy: 0.5628
Epoch 20/20
17/17 [==============================] - 5s 284ms/step - loss: 0.6718 -
accuracy: 0.5969
```

In [30]:
```
loss, acc = full_model.evaluate(ds_test_preprocessed)
print("Accuracy", acc)
```

```
4/4 [==============================] - 1s 59ms/step - loss: 0.6855 - acc
uracy: 0.5000
Accuracy 0.5
```

TODO 4.2.5 Data augmentation introduces noise, continue training and check accuracy

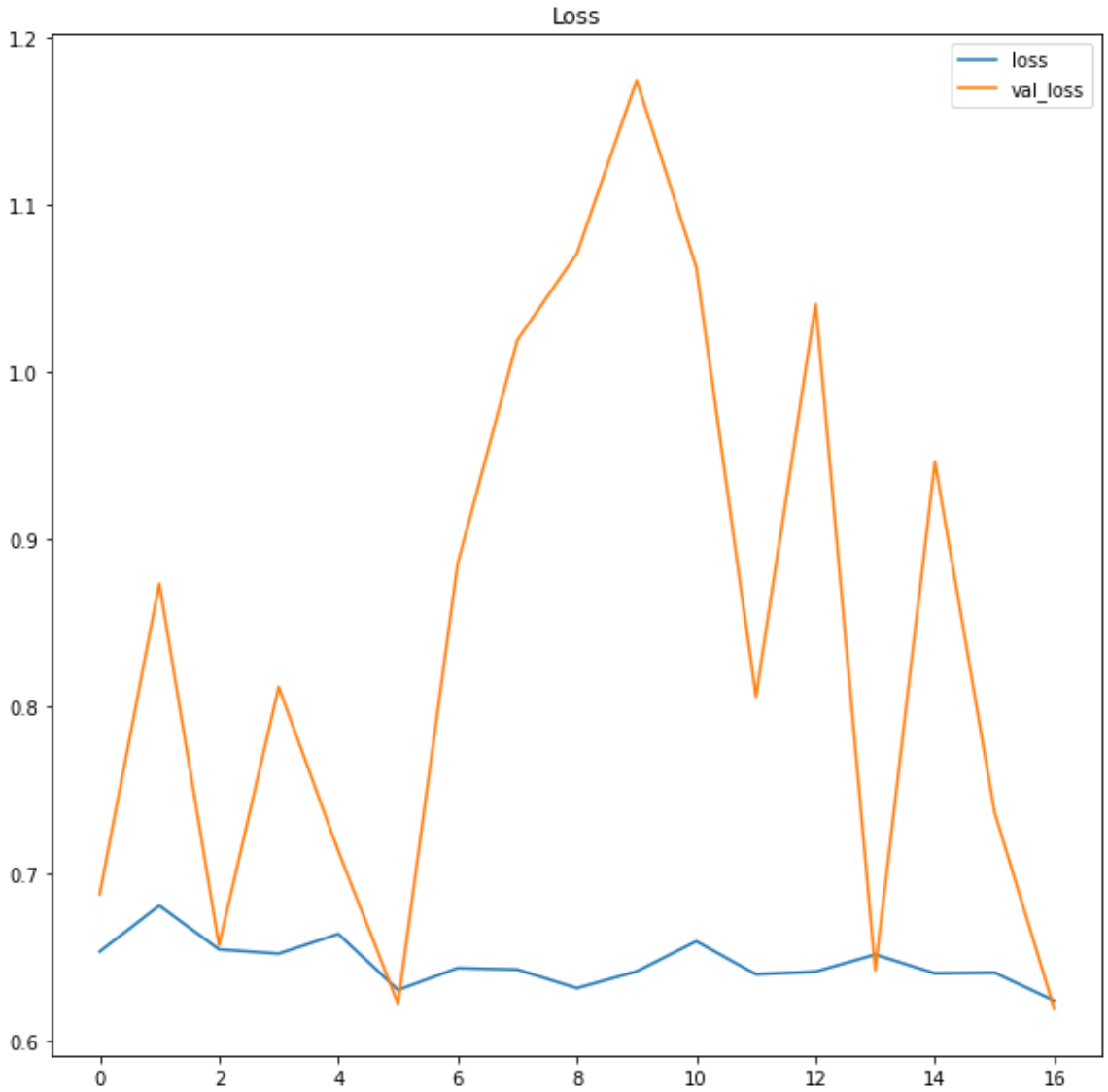
In [34]:
```
epochs=17
hist = full_model.fit(ds_train_preprocessed,epochs=epochs,batch_size=128,
```

```
Epoch 1/17
17/17 [==============================] - 5s 306ms/step - loss: 0.6536 -
accuracy: 0.6290 - val_loss: 0.6879 - val_accuracy: 0.5000
Epoch 2/17
17/17 [==============================] - 5s 313ms/step - loss: 0.6810 -
accuracy: 0.5570 - val_loss: 0.8733 - val_accuracy: 0.5000
Epoch 3/17
17/17 [==============================] - 5s 316ms/step - loss: 0.6549 -
accuracy: 0.6164 - val_loss: 0.6573 - val_accuracy: 0.4844
Epoch 4/17
17/17 [==============================] - 5s 297ms/step - loss: 0.6524 -
accuracy: 0.6047 - val_loss: 0.8117 - val_accuracy: 0.5000
Epoch 5/17
17/17 [==============================] - 5s 318ms/step - loss: 0.6641 -
accuracy: 0.5979 - val_loss: 0.7135 - val_accuracy: 0.5000
Epoch 6/17
17/17 [==============================] - 5s 314ms/step - loss: 0.6307 -
accuracy: 0.6397 - val_loss: 0.6226 - val_accuracy: 0.7500
Epoch 7/17
17/17 [==============================] - 5s 315ms/step - loss: 0.6437 -
accuracy: 0.6397 - val_loss: 0.8852 - val_accuracy: 0.5000
Epoch 8/17
17/17 [==============================] - 5s 295ms/step - loss: 0.6428 -
accuracy: 0.6446 - val_loss: 1.0187 - val_accuracy: 0.5000
Epoch 9/17
17/17 [==============================] - 5s 325ms/step - loss: 0.6318 -
accuracy: 0.6563 - val_loss: 1.0705 - val_accuracy: 0.5000
Epoch 10/17
17/17 [==============================] - 5s 296ms/step - loss: 0.6418 -
accuracy: 0.6290 - val_loss: 1.1739 - val_accuracy: 0.5000
Epoch 11/17
17/17 [==============================] - 5s 308ms/step - loss: 0.6598 -
accuracy: 0.6319 - val_loss: 1.0622 - val_accuracy: 0.5000
Epoch 12/17
17/17 [==============================] - 5s 321ms/step - loss: 0.6400 -
accuracy: 0.6563 - val_loss: 0.8057 - val_accuracy: 0.5000
Epoch 13/17
17/17 [==============================] - 5s 304ms/step - loss: 0.6416 -
accuracy: 0.6251 - val_loss: 1.0404 - val_accuracy: 0.5000
Epoch 14/17
17/17 [==============================] - 5s 322ms/step - loss: 0.6518 -
accuracy: 0.6456 - val_loss: 0.6423 - val_accuracy: 0.5977
Epoch 15/17
17/17 [==============================] - 5s 315ms/step - loss: 0.6406 -
accuracy: 0.6329 - val_loss: 0.9462 - val_accuracy: 0.5000
Epoch 16/17
17/17 [==============================] - 5s 292ms/step - loss: 0.6410 -
accuracy: 0.6475 - val_loss: 0.7367 - val_accuracy: 0.5000
Epoch 17/17
17/17 [==============================] - 5s 305ms/step - loss: 0.6243 -
accuracy: 0.6378 - val_loss: 0.6194 - val_accuracy: 0.7969
```

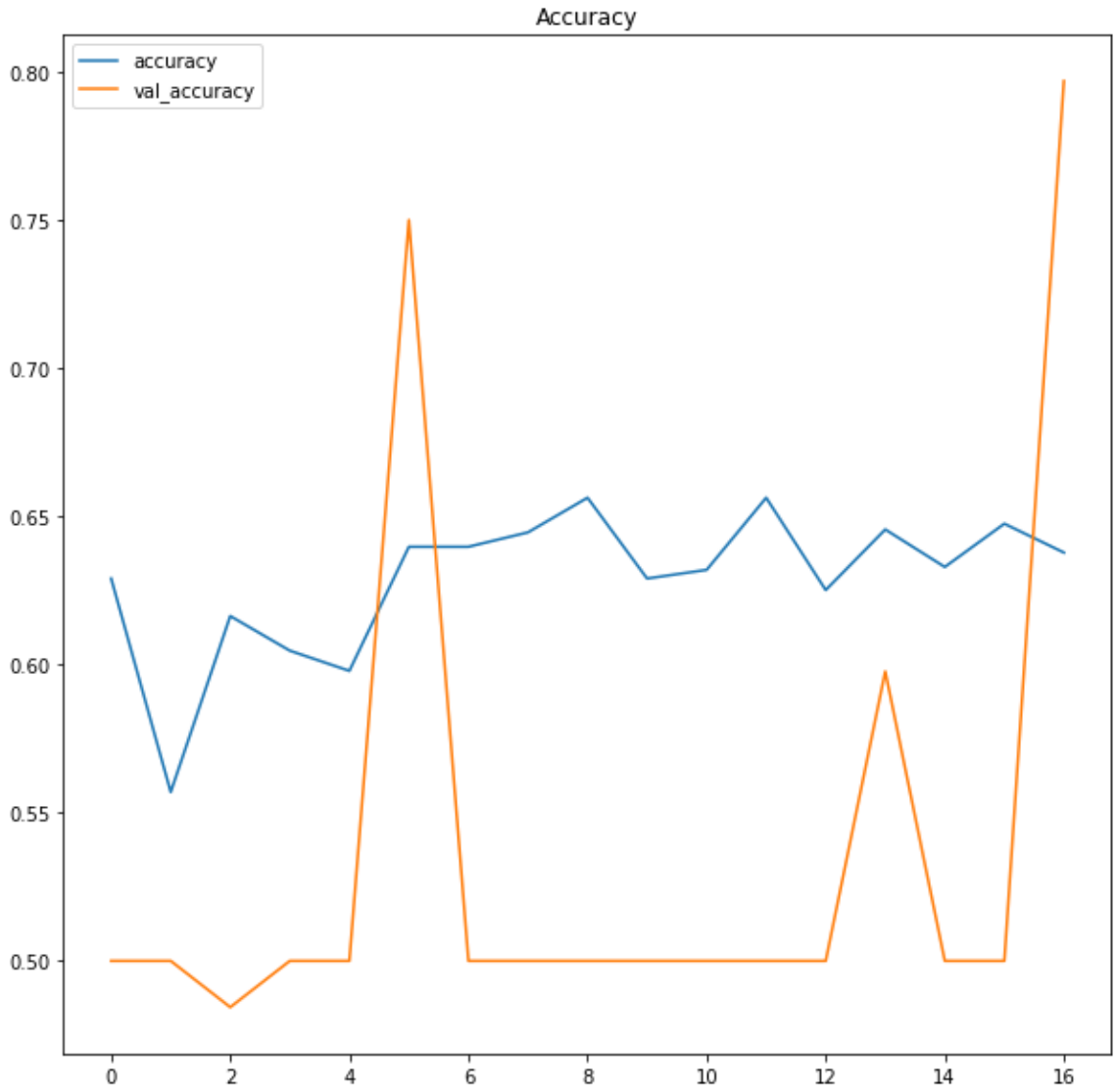
In [35]:
```
loss, acc = full_model.evaluate(ds_test_preprocessed)
print("Accuracy", acc)
```

```
4/4 [==============================] - 0s 58ms/step - loss: 0.6194 - acc
uracy: 0.7969
Accuracy 0.796875
```

In [36]:
```
import matplotlib.pyplot as plt
plt.title('Loss')
plt.plot(hist.history['loss'],label='loss')
plt.plot(hist.history['val_loss'],label='val_loss')
plt.legend()
plt.show()
```



In [37]:
```
plt.title('Accuracy')
plt.plot(hist.history['accuracy'],label='accuracy')
plt.plot(hist.history['val_accuracy'],label='val_accuracy')
plt.legend()
plt.show()
```



4.3 Imagenette

Imagenette is a subset of Imagenet containing dataset of *easy* to classify images. We will use a variant named *imagenette/320px*, which contains images resized to 320 pixels (the shorter side).

Remark: sizes of images in the dataset vary To be delivered to a neural network, images must have fixed dimensions.

In [38]:
```
import tensorflow_datasets as tfds
import tensorflow as tf


ds_train = tfds.load('imagenette/320px', split='train', as_supervised=Tru
ds_test = tfds.load('imagenette/320px', split='validation', as_supervised
```

```
Downloading and preparing dataset Unknown size (download: Unknown size,
generated: Unknown size, total: Unknown size) to /root/tensorflow_datase
ts/imagenette/320px/1.0.0...
Dl Completed...: 0 url [00:00, ? url/s]
Dl Size...: 0 MiB [00:00, ? MiB/s]
Extraction completed...: 0 file [00:00, ? file/s]
```

```
Generating splits...:   0%|          | 0/2 [00:00<?, ? splits/s]
Generating train examples...: 0 examples [00:00, ? examples/s]
Shuffling /root/tensorflow_datasets/imagenette/320px/1.0.0.incompleteVXN
YEO/imagenette-train.tfrecord*...:   0…
Generating validation examples...: 0 examples [00:00, ? examples/s]
Shuffling /root/tensorflow_datasets/imagenette/320px/1.0.0.incompleteVXN
YEO/imagenette-validation.tfrecord*....…
Dataset imagenette downloaded and prepared to /root/tensorflow_datasets/
imagenette/320px/1.0.0. Subsequent calls will reuse this data.
```

In [39]:
```
print(f'Number of images in train set: {ds_train.cardinality()}')
print(f'Number of images in test set: {ds_test.cardinality()}')
```

```
Number of images in train set: 12894
Number of images in test set: 500
```

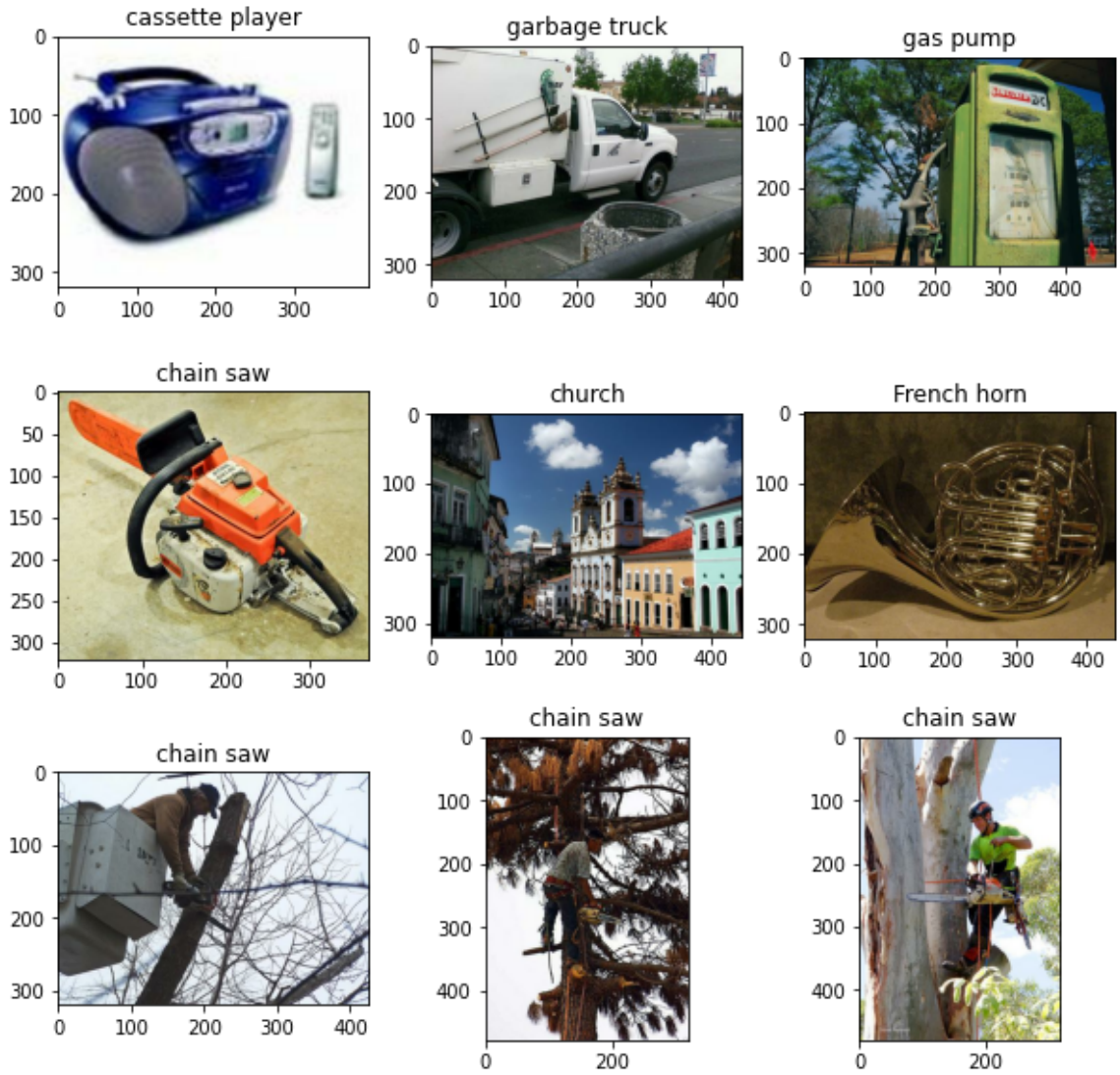
In [40]:
```
labels = [label.numpy() for _, label in ds_train]
print(max(labels)+1)
num_classes = max(labels)+1
```

```
10
```

In [41]:
```
#image sizes?

ds_tmp = ds_train.take(10)
for img,label in ds_tmp:
  print(img.shape)
```

```
(320, 396, 3)
(320, 426, 3)
(320, 477, 3)
(320, 372, 3)
(320, 445, 3)
(320, 440, 3)
(320, 426, 3)
(479, 320, 3)
(480, 320, 3)
(381, 320, 3)
```

In [42]:
```
#display a few images...
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (10,10)
labels=['tench', 'English springer', 'cassette player', 'chain saw', 'chu
        'French horn', 'garbage truck', 'gas pump', 'golf ball', 'parachu
ds2 = ds_train.take(9)
it = ds2.as_numpy_iterator()

input_size=100
for i in range(9):
  ax = plt.subplot(330 + 1 + i)
  image, label = next(it)
  # image = tf.image.resize(image, size=[256,256], method=tf.image.Resize
  ax.set_title(labels[label])
  plt.imshow(image)
plt.show()
```



TODO 4.3.1 Select a transformation that assures equal dimensions of all images (delivered to the neural network). Display images after the transformation.

In [44]:
```
from keras import layers
from keras import models
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
import tensorflow as tf


transform = models.Sequential([
  layers.RandomCrop(height=320, width=320, seed=1),
  layers.Rescaling(1./255)
])
```

In [45]:
```
#display a few images...
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (10,10)

labels=['tench', 'English springer', 'cassette player', 'chain saw', 'chu
        'French horn', 'garbage truck', 'gas pump', 'golf ball', 'parachu

ds2 = ds_train.take(9)
ds2 = ds2.map(lambda x, y: (transform(x), y))
```

```
it = ds2.as_numpy_iterator()

input_size=100
for i in range(9):
  ax = plt.subplot(330 + 1 + i)
  image, label = next(it)
  # image = tf.image.resize(image, size=[256,256], method=tf.image.Resize
  ax.set_title(labels[label])
  plt.imshow(image)
plt.show()
```

4.3.1 CNN classifier

TODO 4.3.2 Define a CNN model comprising several convolution and max pooling layers. As there are 10 classes, the model should have 10 output neurons.

In [47]:
```
from keras import layers
from keras import models
from keras import optimizers

image_width = 320
image_height = 320
```

```
classifier_model = models.Sequential([
    layers.Conv2D(16, (3,3), activation='relu', input_shape=(image_height
    layers.MaxPooling2D(2, 2),
    layers.Dropout(0.1),

    layers.Conv2D(32, (3,3), activation ='relu'),
    layers.MaxPooling2D(2,2),
    layers.Dropout(0.2),

    layers.Conv2D(64, (3,3), activation ='relu'),
    layers.MaxPooling2D(2,2),
    layers.Dropout(0.3),


    layers.Flatten(),

    layers.Dense(512, activation = 'relu'),
    layers.Dense(10, activation = 'sigmoid')
])
```

TODO 4.3.3 Prepare datasets. Assure identical dimensions of images.

In [55]:
```
def prepare_dataset(ds,batch_size=64,resize=None, augment=None):
  if resize is not None:
    ds = ds.map(lambda x, y: (resize(x), y),
                num_parallel_calls=tf.data.AUTOTUNE)

  ds = ds.shuffle(1000)
  ds = ds.batch(batch_size)

  if augment is not None:
    ds = ds.map(lambda x, y: (augment(x), y),
                num_parallel_calls=tf.data.AUTOTUNE)

  return ds.prefetch(buffer_size=tf.data.AUTOTUNE)

ds_train_preprocessed = prepare_dataset(ds_train,batch_size=128,resize=tr
ds_test_preprocessed = prepare_dataset(ds_test,batch_size=128,resize=tran
```

Compile the model

In [49]:
```
classifier_model.compile(loss='sparse_categorical_crossentropy',
              optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),
              metrics=['accuracy'])
classifier_model.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 318, 318, 16)      448

 max_pooling2d_9 (MaxPooling  (None, 159, 159, 16)     0
 2D)

 dropout_9 (Dropout)         (None, 159, 159, 16)      0

 conv2d_10 (Conv2D)          (None, 157, 157, 32)      4640

 max_pooling2d_10 (MaxPoolin  (None, 78, 78, 32)       0
 g2D)

 dropout_10 (Dropout)        (None, 78, 78, 32)        0

 conv2d_11 (Conv2D)          (None, 76, 76, 64)        18496

 max_pooling2d_11 (MaxPoolin  (None, 38, 38, 64)       0
 g2D)

 dropout_11 (Dropout)        (None, 38, 38, 64)        0

 flatten_3 (Flatten)         (None, 92416)             0

 dense_6 (Dense)             (None, 512)               47317504

 dense_7 (Dense)             (None, 10)                5130

=================================================================
Total params: 47,346,218
Trainable params: 47,346,218
Non-trainable params: 0
_________________________________________________________________
```

...and train

In [50]:
```
epochs = 20
hist = classifier_model.fit(ds_train_preprocessed,epochs=epochs,batch_siz
```

```
Epoch 1/20
101/101 [==============================] - 66s 588ms/step - loss: 2.5695
- accuracy: 0.2051 - val_loss: 2.1426 - val_accuracy: 0.2260
Epoch 2/20
101/101 [==============================] - 59s 566ms/step - loss: 1.9186
- accuracy: 0.3396 - val_loss: 1.8828 - val_accuracy: 0.3980
Epoch 3/20
101/101 [==============================] - 58s 552ms/step - loss: 1.6976
- accuracy: 0.4307 - val_loss: 1.7264 - val_accuracy: 0.4680
Epoch 4/20
101/101 [==============================] - 59s 557ms/step - loss: 1.5592
- accuracy: 0.4832 - val_loss: 1.6370 - val_accuracy: 0.4800
Epoch 5/20
101/101 [==============================] - 59s 567ms/step - loss: 1.4548
- accuracy: 0.5163 - val_loss: 1.5417 - val_accuracy: 0.5220
Epoch 6/20
101/101 [==============================] - 57s 547ms/step - loss: 1.3638
- accuracy: 0.5515 - val_loss: 1.4997 - val_accuracy: 0.5400
Epoch 7/20
101/101 [==============================] - 58s 551ms/step - loss: 1.2944
- accuracy: 0.5802 - val_loss: 1.4836 - val_accuracy: 0.5280
Epoch 8/20
101/101 [==============================] - 59s 561ms/step - loss: 1.2296
- accuracy: 0.6034 - val_loss: 1.4483 - val_accuracy: 0.5060
Epoch 9/20
101/101 [==============================] - 58s 552ms/step - loss: 1.1720
- accuracy: 0.6216 - val_loss: 1.2990 - val_accuracy: 0.5920
Epoch 10/20
101/101 [==============================] - 60s 560ms/step - loss: 1.1128
- accuracy: 0.6420 - val_loss: 1.3340 - val_accuracy: 0.5540
Epoch 11/20
101/101 [==============================] - 58s 560ms/step - loss: 1.0592
- accuracy: 0.6588 - val_loss: 1.3668 - val_accuracy: 0.5440
Epoch 12/20
101/101 [==============================] - 58s 559ms/step - loss: 1.0124
- accuracy: 0.6757 - val_loss: 1.3683 - val_accuracy: 0.5460
Epoch 13/20
101/101 [==============================] - 58s 562ms/step - loss: 0.9595
- accuracy: 0.6984 - val_loss: 1.2051 - val_accuracy: 0.5980
Epoch 14/20
101/101 [==============================] - 57s 549ms/step - loss: 0.9148
- accuracy: 0.7078 - val_loss: 1.2085 - val_accuracy: 0.5920
Epoch 15/20
101/101 [==============================] - 58s 554ms/step - loss: 0.8748
- accuracy: 0.7219 - val_loss: 1.1520 - val_accuracy: 0.6240
Epoch 16/20
101/101 [==============================] - 59s 559ms/step - loss: 0.8190
- accuracy: 0.7407 - val_loss: 1.1279 - val_accuracy: 0.6320
Epoch 17/20
101/101 [==============================] - 58s 549ms/step - loss: 0.7837
- accuracy: 0.7528 - val_loss: 1.1329 - val_accuracy: 0.6080
Epoch 18/20
101/101 [==============================] - 60s 573ms/step - loss: 0.7314
- accuracy: 0.7657 - val_loss: 1.1410 - val_accuracy: 0.6260
Epoch 19/20
101/101 [==============================] - 58s 553ms/step - loss: 0.6886
- accuracy: 0.7858 - val_loss: 1.1700 - val_accuracy: 0.6060
Epoch 20/20
101/101 [==============================] - 59s 557ms/step - loss: 0.6476
- accuracy: 0.7974 - val_loss: 1.2797 - val_accuracy: 0.5700
```

TODO 4.3.3 Compute accuracy

In [51]:
```
loss, acc = classifier_model.evaluate(ds_test_preprocessed)
print("Accuracy", acc)
```

```
4/4 [==============================] - 2s 124ms/step - loss: 1.2797 - ac
curacy: 0.5700
Accuracy 0.5699999928474426
```

In [52]:
```
classifier_model.save('imagenette_cnn.h5')
```

4.3.2 Transfer learning

- We will use pretrained vgg16 model for **feature extraction**.
- Than we will flatten features to a vector form and add hidden dense layer
- Finally, size of the output layer will correspond to the number of classes

In [53]:
```
from keras.layers import Conv2D, Dense, Flatten, MaxPooling2D
from keras import Sequential

from keras.applications.vgg16 import VGG16
from keras.models import Model

def define_vgg16_model():
  # load model
  model = VGG16(include_top=False, input_shape=(320, 320, 3))
  # mark loaded layers as not trainable
  for layer in model.layers:
    layer.trainable = False
  # add new classifier layers
  full_model = models.Sequential([
      model,
      Flatten(),
      Dense(128, activation='relu', kernel_initializer='he_uniform'),
      Dense(num_classes, activation='softmax')
      ])

  full_model.compile(optimizer='adam', loss='sparse_categorical_crossentr

  return full_model

model = define_vgg16_model()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-ap
plications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 3s 0us/step
58900480/58889256 [==============================] - 3s 0us/step
```

TODO 4.3.4 Train the model and compute accuracy on the test data

In [57]:
```
epochs = 2
hist = model.fit(ds_train_preprocessed, epochs=epochs, batch_size=128, va
```

```
Epoch 1/2
101/101 [==============================] - 239s 2s/step - loss: 1.0298 -
accuracy: 0.7489 - val_loss: 0.3145 - val_accuracy: 0.9280
Epoch 2/2
101/101 [==============================] - 181s 2s/step - loss: 0.2673 -
accuracy: 0.9268 - val_loss: 0.2824 - val_accuracy: 0.9080
```

In [58]:
```
loss, acc = model.evaluate(ds_test_preprocessed)
print("Accuracy", acc) #should be over 90%
```

```
4/4 [==============================] - 7s 1s/step - loss: 0.2824 - accur
acy: 0.9080
Accuracy 0.9079999923706055
```

In [59]:
```
model.save('imaginette_vgg16.h5')
```

Compute scores and confusion matrix

We need true labels (y_test) and predictions (y_pred)

In [60]:
```
y_test=[label.numpy() for img,label in ds_test]
y_test=np.array(y_test)
y_test.shape
```

Out[60]:
```
(500,)
```

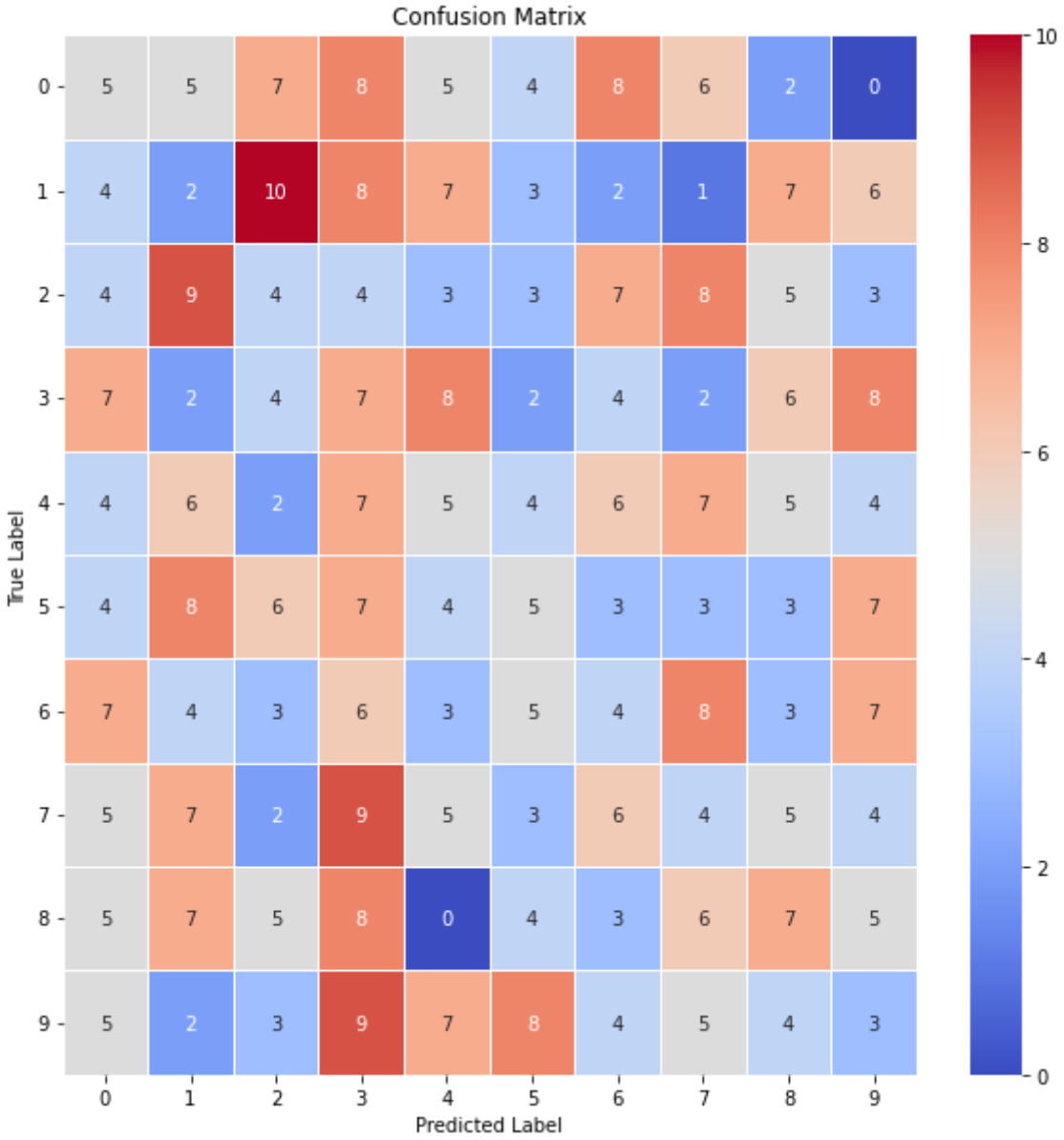
In [61]:
```
import numpy as np
probs = model.predict(ds_test_preprocessed)
print(f'{probs.shape}')
y_pred = np.argmax(probs,axis=1)
```

```
(500, 10)
```

In [62]:
```
import seaborn as sns
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt


# Define the confusion matrix for the results
def show_confusion_matrix(matrix, labels=None):
    num_classes = matrix.shape[0]
    if labels is None:
      labels = [str(i) for i in range(num_classes)]
    plt.figure(figsize=(num_classes, num_classes))
    hm = sns.heatmap(matrix,
                cmap='coolwarm',
                linecolor='white',
                linewidths=1,
                xticklabels=labels[0:num_classes],
                yticklabels=labels[0:num_classes],
                annot=True,
                fmt='d')
    plt.yticks(rotation = 0)  # Don't rotate (vertically) the y-axis labe
    # hm.set_ylim(0, len(matrix))
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()
```

```
def compute_and_show_confusion_matrix(validations, predictions, labels=No
    matrix = metrics.confusion_matrix(validations, predictions)
    show_confusion_matrix(matrix, labels)
```

In [63]:
```
compute_and_show_confusion_matrix(y_test,y_pred)
print(classification_report(y_test, y_pred))
```



	precision	recall	f1-score	support
0	0.10	0.10	0.10	50
1	0.04	0.04	0.04	50
2	0.09	0.08	0.08	50
3	0.10	0.14	0.11	50
4	0.11	0.10	0.10	50
5	0.12	0.10	0.11	50
6	0.09	0.08	0.08	50
7	0.08	0.08	0.08	50
8	0.15	0.14	0.14	50
9	0.06	0.06	0.06	50
accuracy			0.09	500
macro avg	0.09	0.09	0.09	500
weighted avg	0.09	0.09	0.09	500

Soemthing went wrong?

Possible cause - ds_test_preprocessed stream contains **shuffle()** stage.

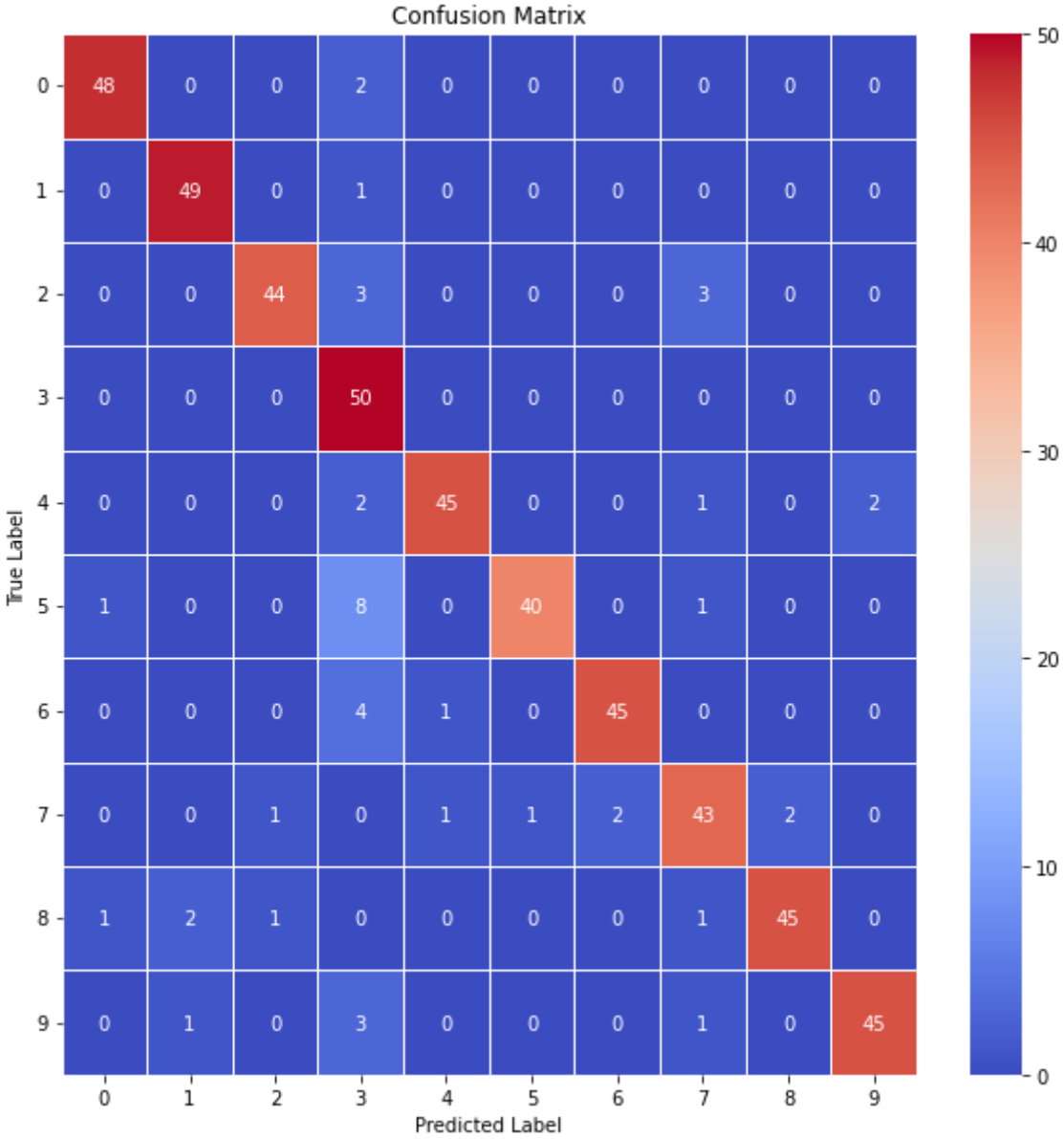
TODO 4.3.5 prepare a Dataset `ds_test_validation`, which does not involve reoredring images (but transforms them to correct size and correctly prepare batches). Then repeat evaluation steps.

In [64]:
```
ds_test_validation = ds_test.map(lambda x, y: (transform(x), y)).batch(12
```

In [65]:
```
#compute y_pred using ds_test_validation

probs = model.predict(ds_test_validation)
print(f'{probs.shape}')
y_pred = np.argmax(probs,axis=1)
```

```
(500, 10)
```

In [66]:
```
compute_and_show_confusion_matrix(y_test,y_pred)
print(classification_report(y_test, y_pred))
```



	precision	recall	f1-score	support
0	0.96	0.96	0.96	50
1	0.94	0.98	0.96	50
2	0.96	0.88	0.92	50
3	0.68	1.00	0.81	50
4	0.96	0.90	0.93	50
5	0.98	0.80	0.88	50
6	0.96	0.90	0.93	50
7	0.86	0.86	0.86	50
8	0.96	0.90	0.93	50
9	0.96	0.90	0.93	50
accuracy			0.91	500
macro avg	0.92	0.91	0.91	500
weighted avg	0.92	0.91	0.91	500