Cifar10

Passons à autre chose que du Cats & Dogs. Nous allons essayer de créer un CNN qui va classer non pas 2 mais 10 labels d'images différents. Nous tirons les images du dataset nommé Cifar10

- Importez :
  - Tensorflow 2.0
  - Numpy
  - Matplotlib.pyplot

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

- En utilisant `tf.keras.datasets` importez cifar10 et stockez les données dans `X_train, y_train, X_test, y_test`

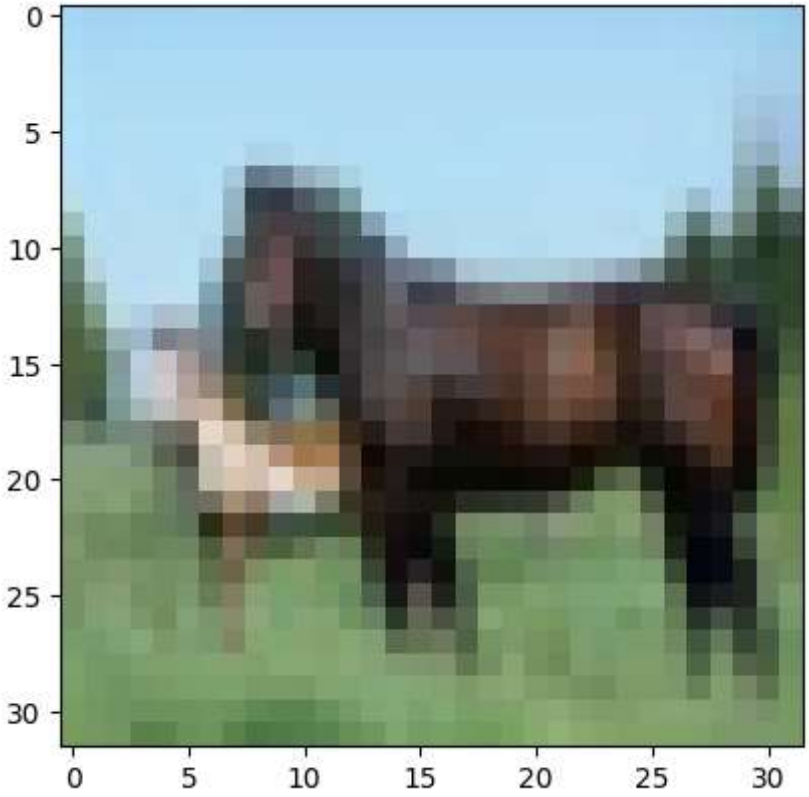
In [2]:
```
(x_train,y_train),(x_test,y_test)=keras.datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 6s 0us/step
```

- Visualisez une image aléatoire de votre dataset

In [8]:
```
# On génére un nombre aléatoire entre 0 et 100
x = np.random.randint(0,50)

# On affiche l'image N°'x'
plt.imshow(x_train[x])
plt.show()
```

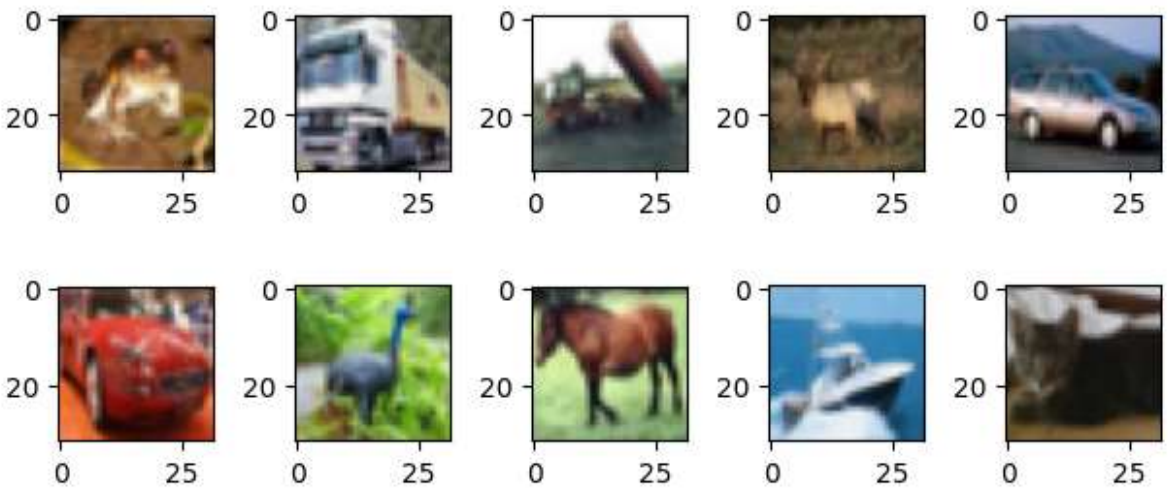


- Visualisez une image aléatoire de chacun de vos labels dans votre dataset

In [9]:

```
num = 10
images = x_train[:num]
labels = y_train[:num]
num_row = 2
num_col = 5

# plot images
fig, axes = plt.subplots(num_row , num_col, figsize=(1.25*num_col , 1.5*num_row))
for i in range(num):
    ax = axes[i//num_col , i%num_col]
    ax.imshow(images[i] , cmap = 'gray')
plt.tight_layout()
plt.show()
```



- Stockez vos images et labels d'entrainements dans un `tf.data.Dataset`
- Faites de mêmes avec les images et labels de validation

In [10]:
```
ds_train = tf.data.Dataset.from_tensor_slices((x_train, y_train))
ds_valid = tf.data.Dataset.from_tensor_slices((x_test, y_test))
```

- Mélangez votre dataset via `.shuffle()` et créez des batchs de 16 images via `.batch()`

In [11]:
```
ds_train = ds_train.shuffle(50000).batch(16)
ds_valid = ds_valid.shuffle(50000).batch(16)
```

- Regardez un exemple

In [12]:
```
for image in ds_train.take(1):
  print(image)
```

```
(<tf.Tensor: shape=(16, 32, 32, 3), dtype=uint8, numpy=
array([[[[ 35,  55, 105],
         [ 35,  55, 105],
         [ 34,  54, 104],
         ...,
         [ 57,  55,  94],
         [ 55,  53,  95],
         [ 49,  50,  89]],

        [[ 37,  57, 107],
         [ 42,  62, 112],
         [ 39,  59, 109],
         ...,
         [116, 114, 141],
         [101,  99, 130],
         [ 70,  71, 104]],

        [[ 37,  57, 107],
         [ 44,  64, 114],
         [ 42,  62, 112],
         ...,
         [208, 205, 221],
         [216, 215, 230],
         [165, 167, 190]],

        ...,

        [[248, 246, 252],
         [253, 248, 253],
         [251, 242, 247],
         ...,
         [ 10,   4,  20],
         [ 28,  23,  41],
         [ 29,  26,  47]],

        [[250, 248, 253],
         [252, 248, 253],
         [250, 241, 248],
         ...,
         [  6,   0,  16],
         [ 19,  13,  32],
         [ 41,  38,  59]],

        [[243, 242, 248],
         [248, 243, 251],
         [247, 237, 249],
         ...,
         [  7,   0,  18],
         [ 10,   5,  23],
         [ 31,  27,  49]]],


       [[[128, 142, 161],
         [106, 120, 131],
         [ 28,  35,  42],
         ...,
         [186, 188, 166],
         [185, 182, 155],
         [147, 144, 118]],

        [[130, 143, 162],
         [119, 133, 144],
         [ 54,  62,  68],
         ...,
```

```
  [205, 205, 190],
  [229, 223, 204],
  [203, 197, 171]],

 [[139, 152, 171],
  [139, 153, 164],
  [104, 111, 118],
  ...,
  [116, 127, 124],
  [137, 143, 138],
  [136, 143, 130]],

 ...,

 [[ 94, 105, 109],
  [ 94, 106, 106],
  [107, 120, 115],
  ...,
  [108, 122, 125],
  [115, 129, 132],
  [107, 121, 124]],

 [[102, 113, 116],
  [ 98, 110, 110],
  [106, 118, 116],
  ...,
  [101, 115, 118],
  [118, 132, 135],
  [115, 129, 132]],

 [[106, 117, 119],
  [ 99, 110, 111],
  [103, 115, 115],
  ...,
  [105, 118, 121],
  [110, 124, 127],
  [113, 127, 130]]],


[[[ 76, 113,  68],
  [ 73, 112,  71],
  [ 58, 110,  57],
  ...,
  [101, 197, 107],
  [101, 196,  91],
  [ 96, 182,  78]],

 [[ 61,  97,  48],
  [ 66, 106,  65],
  [ 62, 103,  60],
  ...,
  [ 98, 192, 102],
  [102, 187,  87],
  [102, 167,  71]],

 [[ 50,  86,  36],
  [ 60, 103,  52],
  [ 61,  93,  51],
  ...,
  [101, 185,  98],
  [101, 178,  80],
  [109, 183,  84]],

 ...,
```

```
 [[138, 139,  88],
  [139, 132, 107],
  [133, 124, 110],
  ...,
  [ 85,  89,  55],
  [102, 101,  58],
  [ 99,  96,  64]],

 [[140, 152, 104],
  [140, 131, 103],
  [159, 143, 130],
  ...,
  [110, 126,  82],
  [104, 102,  63],
  [100,  96,  68]],

 [[157, 160, 123],
  [140, 136, 103],
  [137, 131, 108],
  ...,
  [110, 131,  74],
  [129, 127,  91],
  [114, 110,  85]]],


...,


[[[152, 125,  87],
  [161, 129,  91],
  [163, 126,  91],
  ...,
  [169, 124,  92],
  [186, 141, 102],
  [202, 158, 116]],

 [[147, 120,  85],
  [164, 131,  93],
  [168, 132,  95],
  ...,
  [172, 128,  94],
  [203, 157, 115],
  [221, 178, 133]],

 [[139, 111,  79],
  [163, 129,  93],
  [172, 136,  97],
  ...,
  [174, 129,  95],
  [209, 162, 120],
  [220, 180, 135]],

 ...,

 [[163, 145,  90],
  [161, 137,  89],
  [196, 168, 120],
  ...,
  [ 71,  68,  44],
  [ 62,  61,  42],
  [ 83,  77,  50]],

 [[161, 138,  82],
```

```
  [137, 119,  75],
  [173, 150, 106],
  ...,
  [ 68,  66,  39],
  [ 59,  58,  35],
  [108,  91,  55]],

 [[164, 138,  87],
  [133, 114,  75],
  [156, 136,  96],
  ...,
  [122, 114,  88],
  [ 76,  74,  53],
  [122, 102,  60]]],


[[[  0,   5,   0],
  [  0,   5,   0],
  [  0,   5,   0],
  ...,
  [ 87,  55,  32],
  [ 86,  54,  31],
  [ 86,  56,  35]],

 [[  1,   6,   1],
  [  1,   6,   1],
  [  1,   6,   0],
  ...,
  [ 87,  54,  31],
  [ 87,  55,  31],
  [ 91,  60,  39]],

 [[  2,   7,   1],
  [  2,   7,   1],
  [  2,   7,   1],
  ...,
  [ 86,  54,  31],
  [ 89,  57,  34],
  [ 95,  65,  43]],

 ...,

 [[165, 143, 127],
  [159, 140, 122],
  [163, 147, 129],
  ...,
  [100,  82,  78],
  [111,  93,  77],
  [ 82,  64,  42]],

 [[153, 130, 114],
  [156, 137, 120],
  [164, 148, 129],
  ...,
  [108,  88,  82],
  [105,  85,  70],
  [ 66,  46,  28]],

 [[129, 107,  91],
  [152, 133, 116],
  [159, 143, 125],
  ...,
  [106,  86,  77],
  [ 89,  67,  53],
```

```
          [ 62,  41,  26]]],


        [[[ 91,  59,  42],
          [ 69,  35,  19],
          [ 39,  14,   4],
          ...,
          [171, 141, 109],
          [170, 149, 118],
          [183, 164, 145]],

         [[100,  64,  44],
          [ 64,  33,  19],
          [ 38,  15,   5],
          ...,
          [171, 140, 109],
          [171, 149, 119],
          [185, 166, 147]],

         [[100,  66,  45],
          [ 61,  32,  21],
          [ 37,  12,   4],
          ...,
          [173, 140, 110],
          [173, 150, 120],
          [186, 165, 147]],

         ...,

         [[ 94, 116, 144],
          [102, 121, 152],
          [ 97, 116, 150],
          ...,
          [104, 135, 171],
          [104, 134, 170],
          [107, 136, 171]],

         [[ 84, 108, 141],
          [ 93, 114, 149],
          [ 98, 119, 155],
          ...,
          [ 96, 127, 163],
          [ 96, 128, 164],
          [101, 135, 172]],

         [[ 80, 103, 142],
          [ 87, 108, 145],
          [103, 124, 161],
          ...,
          [107, 134, 167],
          [ 96, 128, 164],
          [ 91, 129, 169]]]], dtype=uint8)>, <tf.Tensor: shape=(16, 1), dtype=uint
8, numpy=
array([[8],
       [5],
       [6],
       [1],
       [7],
       [2],
       [7],
       [2],
       [6],
       [8],
       [1],
```

```
       [0],
       [7],
       [2],
       [3],
       [1]], dtype=uint8)>)
```

- Créez un modèle CNN dans lequel vous mettrez des couches convolutionnelles 2D et des couches MaxPool2D

In [13]:
```
model = keras.models.Sequential()

model.add( keras.layers.Input((32, 32, 3)) )

model.add( keras.layers.Conv2D(16, (3,3),  activation='relu') )
model.add(keras.layers.BatchNormalization(axis=-1))
model.add( keras.layers.Conv2D(32, (3,3), activation='relu') )
model.add( keras.layers.MaxPooling2D((2,2)))
model.add( keras.layers.Conv2D(64, (3,3),  activation='relu') )
model.add(keras.layers.BatchNormalization(axis=-1))
model.add( keras.layers.Conv2D(32, (3,3), activation='relu') )
model.add( keras.layers.MaxPooling2D((2,2)))

model.add( keras.layers.Flatten())
model.add( keras.layers.Dense(512, activation='relu'))
model.add( keras.layers.Dense(10, activation='softmax'))
```

- Créez une *Learning Rate Schedule*. Vous pouvez choisir de le faire avec *ExponentialDecay* ou d'autres méthodes

In [14]:
```
initial_learning_rate = 0.1
lr_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=100000,
    decay_rate=0.96,
    staircase=True)
```

- Créez un compileur dans lequel vous choisirez :
  - Votre optimiseur
  - Votre loss
  - Vos métrics

In [15]:
```
model.compile(optimizer=keras.optimizers.SGD(learning_rate=lr_schedule),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

- Faites tourner votre modèle sur 10 epochs

In [16]:
```
#epochs= 1
history = model.fit( ds_train,
                     epochs = 10)
```

```
Epoch 1/10
3125/3125 [==============================] - 25s 5ms/step - loss: 1.4610 - accurac
y: 0.4811
Epoch 2/10
3125/3125 [==============================] - 16s 5ms/step - loss: 1.0409 - accurac
y: 0.6379
Epoch 3/10
3125/3125 [==============================] - 16s 5ms/step - loss: 0.8774 - accurac
y: 0.6979
Epoch 4/10
3125/3125 [==============================] - 22s 7ms/step - loss: 0.7663 - accurac
y: 0.7334
Epoch 5/10
3125/3125 [==============================] - 15s 5ms/step - loss: 0.6753 - accurac
y: 0.7672
Epoch 6/10
3125/3125 [==============================] - 15s 5ms/step - loss: 0.5991 - accurac
y: 0.7948
Epoch 7/10
3125/3125 [==============================] - 16s 5ms/step - loss: 0.5309 - accurac
y: 0.8154
Epoch 8/10
3125/3125 [==============================] - 16s 5ms/step - loss: 0.4874 - accurac
y: 0.8331
Epoch 9/10
3125/3125 [==============================] - 15s 5ms/step - loss: 0.4425 - accurac
y: 0.8502
Epoch 10/10
3125/3125 [==============================] - 15s 5ms/step - loss: 0.4032 - accurac
y: 0.8647
```

- Evaluez votre modèle

In [17]:

```
# Generate generalization metrics
score = model.evaluate(x_test, y_test)
print(f'test loss: {score[0]} / test accuracy: {score[1]}')
```

```
313/313 [==============================] - 1s 3ms/step - loss: 1.1717 - accuracy:
0.7055
test loss: 1.1716597080230713 / test accuracy: 0.7055000066757202
```

- Tentez d'entrainer votre modèle à nouveau. Cette fois cependant :
  - Ajoutez l'argument *validation_data = ds_valid*
  - Profitez en pour enregistrer votre entrainement dans une variable *history*
- Que pouvez vous conclure sur votre modèle ?

In [18]:

```
history = model.fit( ds_train,
                     epochs = 10,
                     validation_data=ds_valid)
```

```
Epoch 1/10
3125/3125 [==============================] - 17s 5ms/step - loss: 0.3892 - accurac
y: 0.8714 - val_loss: 1.2350 - val_accuracy: 0.6936
Epoch 2/10
3125/3125 [==============================] - 17s 5ms/step - loss: 0.3740 - accurac
y: 0.8761 - val_loss: 1.3275 - val_accuracy: 0.6923
Epoch 3/10
3125/3125 [==============================] - 17s 5ms/step - loss: 0.3521 - accurac
y: 0.8868 - val_loss: 1.4022 - val_accuracy: 0.6897
Epoch 4/10
3125/3125 [==============================] - 20s 6ms/step - loss: 0.3392 - accurac
y: 0.8926 - val_loss: 1.5825 - val_accuracy: 0.6776
Epoch 5/10
3125/3125 [==============================] - 17s 5ms/step - loss: 0.3468 - accurac
y: 0.8923 - val_loss: 1.6889 - val_accuracy: 0.6819
Epoch 6/10
3125/3125 [==============================] - 19s 6ms/step - loss: 0.3435 - accurac
y: 0.8945 - val_loss: 1.6029 - val_accuracy: 0.6966
Epoch 7/10
3125/3125 [==============================] - 17s 5ms/step - loss: 0.3242 - accurac
y: 0.9031 - val_loss: 1.6157 - val_accuracy: 0.6930
Epoch 8/10
3125/3125 [==============================] - 17s 6ms/step - loss: 0.3348 - accurac
y: 0.9009 - val_loss: 1.9719 - val_accuracy: 0.6565
Epoch 9/10
3125/3125 [==============================] - 17s 5ms/step - loss: 0.3409 - accurac
y: 0.8982 - val_loss: 1.9995 - val_accuracy: 0.6632
Epoch 10/10
3125/3125 [==============================] - 18s 6ms/step - loss: 0.3425 - accurac
y: 0.9033 - val_loss: 1.9554 - val_accuracy: 0.6675
```

- Regardez ce qu'il y a dans `history.history`

In [19]:

```
history.history
```

Out[19]:

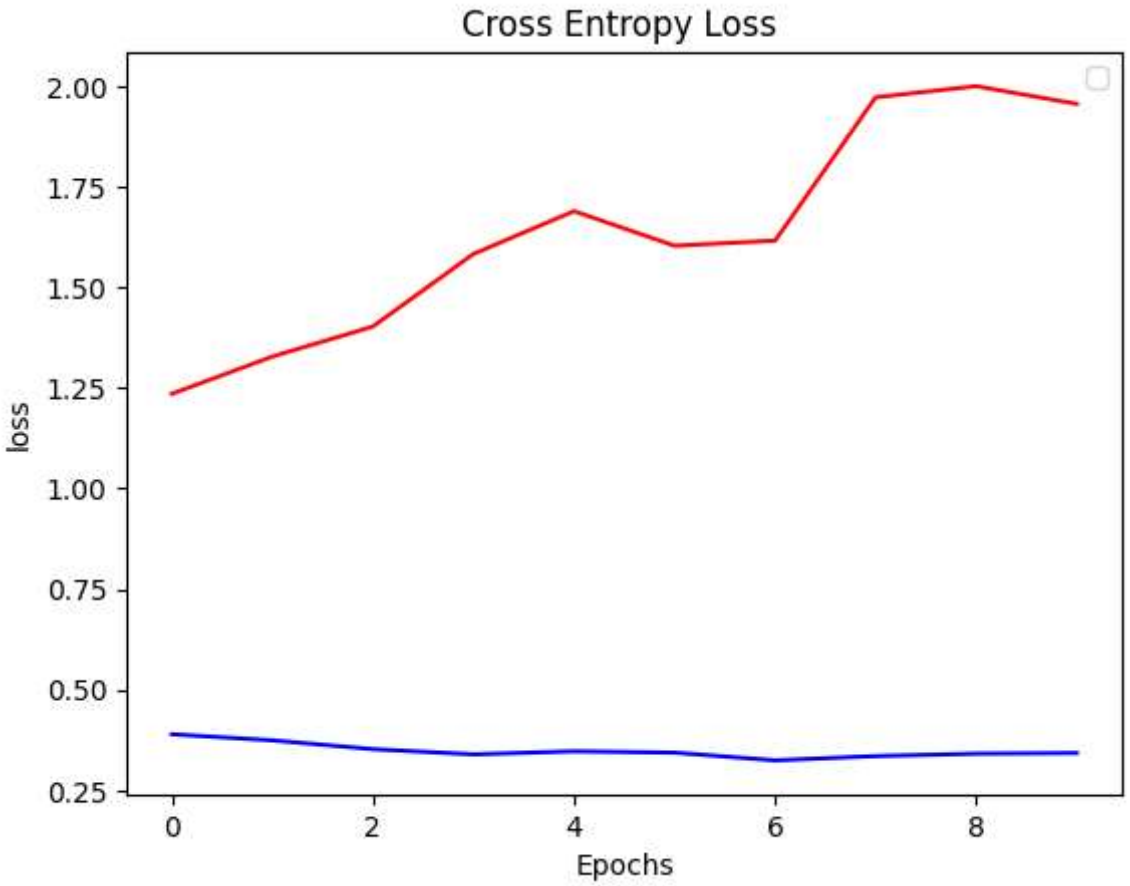
```
{'loss': [0.3891778588294983,
  0.37400221824645996,
  0.35209840536117554,
  0.33916881680488586,
  0.34680742025375366,
  0.34346210956573486,
  0.32423073053359985,
  0.33484241366386414,
  0.3409419655799866,
  0.34249186515808105],
 'accuracy': [0.8713799715042114,
  0.8761199712753296,
  0.8868200182914734,
  0.8926399946212769,
  0.8923400044441223,
  0.894540011882782,
  0.9031199812889099,
  0.9009400010108948,
  0.8982200026512146,
  0.9032999873161316],
 'val_loss': [1.2349879741668701,
  1.3274673223495483,
  1.4022197723388672,
  1.5825252532958984,
  1.6889312267303467,
  1.602907657623291,
  1.6156936883926392,
  1.9718546867370605,
  1.999501347541809,
  1.9553570747375488],
 'val_accuracy': [0.6935999989509583,
  0.692300021648407,
  0.6897000074386597,
  0.6776000261306763,
  0.6819000244140625,
  0.6966000199317932,
  0.6930000185966492,
  0.656499981880188,
  0.6632000207901001,
  0.6675000190734863]}
```


- Regardez maintenant un graphique de l'évolution de votre *loss* à mesure que les epochs avancent

In [20]:
```
plt.subplot()
plt.title('Cross Entropy Loss ')
plt.plot(history.history['loss'], color='blue' )
plt.plot(history.history['val_loss'], color='red')
plt.xlabel('Epochs')
plt.ylabel('loss')
plt.legend()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note tha
t artists whose label start with an underscore are ignored when legend() is called
with no argument.
```

Out[20]:
```
<matplotlib.legend.Legend at 0x7fc5507d3f70>
```



- Faites de même avec votre Accuracy

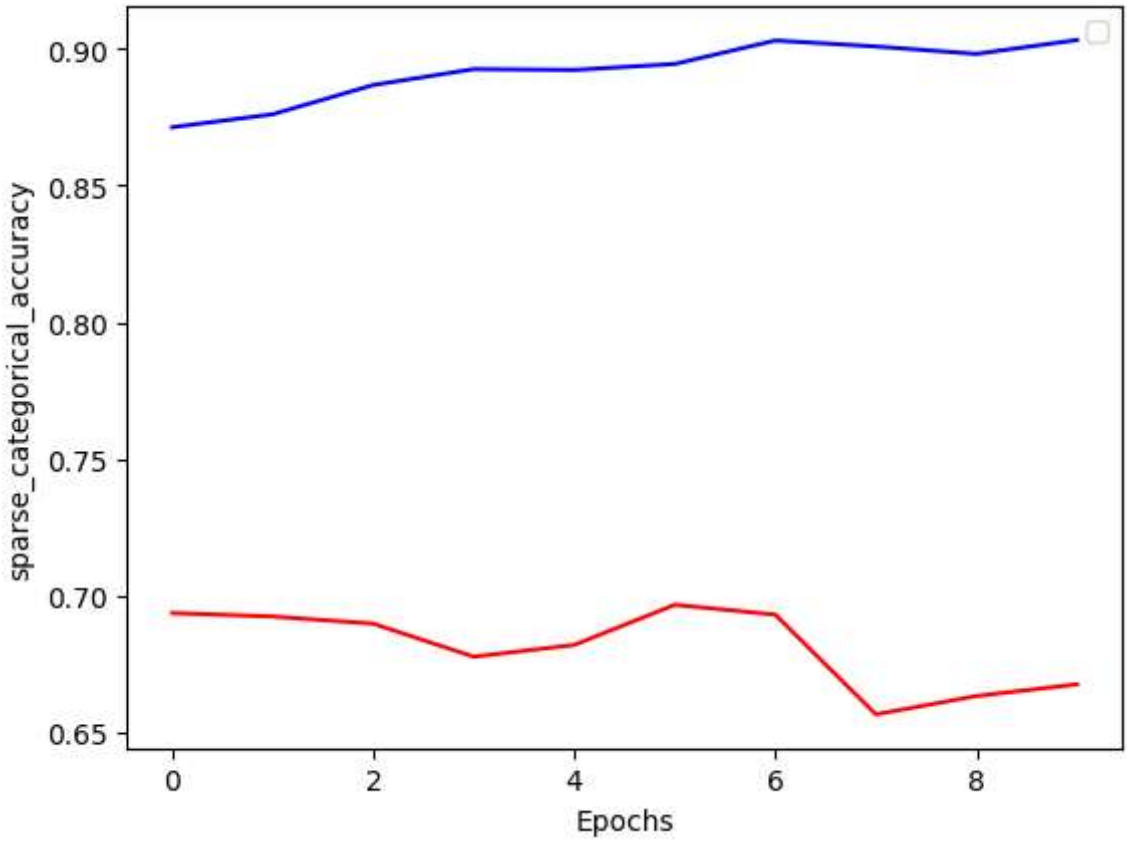
In [21]:

```
plt.subplot()
plt.plot(history.history['accuracy'], color='blue')
plt.plot(history.history['val_accuracy'], color='red')
plt.xlabel('Epochs')
plt.ylabel('sparse_categorical_accuracy')
plt.legend()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note tha
t artists whose label start with an underscore are ignored when legend() is called
with no argument.
```

Out[21]:

```
<matplotlib.legend.Legend at 0x7fc5507f67a0>
```



- Un moyen de gérer l'overfitting est de faire ce qu'on appelle de la *Data Augementation*. Le principe est de prendre les images que nous avons dans le jeu de données et de les répéter avec quelques changement mineurs comme :
  - Changer la couleur, la luminosité
  - Retourner l'image
  - Couper l'image

Cette technique permet de créer de nouvelles images sur lesquelles le modèle peut s'entrainer et donc mieux performer ! Tentons de voir ce que cela pourrait donner sur notre jeu de données.

- Commencez par utiliser la fonction `.unbatch()` sur votre jeu de données d'entrainement et de validation. Nous allons changer la taille des batchs plus tard

In [22]:
```
ds_train = ds_train.unbatch()
ds_valid = ds_valid.unbatch()
```

- Vérifiez que vous ayez bien des tenseurs de taille (32, 32, 3) en regardant un exemple dans votre dataset

In [23]:
```
ds_train
```

Out[23]:
```
<_UnbatchDataset element_spec=(TensorSpec(shape=(32, 32, 3), dtype=tf.uint8, name=
None), TensorSpec(shape=(1,), dtype=tf.uint8, name=None))>
```

- En utilisant la fonction `repeat()` copiez votre dataset 10 fois

In [24]:
```
ds_train = ds_train.repeat(10)
```

- On va procéder maintenant à une phase de Data Augmentation. Créez une fonction `data_aug(image, label)` qui prendra comme argument une image et un label. A l'intérieur de cette fonction, tentez d'utiliser :
  - tf.image.random_flip_left_right
  - tf.image.random_contrast
  - tf.image.random_crop
  - Divisez le tenseur par 255
  - Vous pouvez tentez d'autres choses si vous le souhaitez

In [25]:
```
def data_aug(img,label):
  img = tf.image.random_flip_left_right(img)
  img = tf.image.random_contrast(img, 0.50, 0.90)
  img = tf.image.random_crop(img,(16,16,3))
  img = tf.image.resize(img ,[32,32])
  img = img / 255.0
  return img , label
```

- Utilisez la fonction `.map()` pour appliquer votre fonction sur votre tf dataset

In [26]:
```
ds_aug = ds_train.map(lambda x, y: (data_aug(x, y)))
```

- Créez une fonction pour votre jeu de données de validation où vous diviserez simplement les tenseurs par 255. Nous ne voulons pas appliquer de data augmentation sur notre jeu de données de validation

In [27]:
```
def division(img ,label):
  img = img/255
  return img , label
```

- Appliquez la fonction sur votre jeu de données de validation

In [28]:
```
ds_valid = ds_valid.map(lambda x , y: division(x ,y))
```

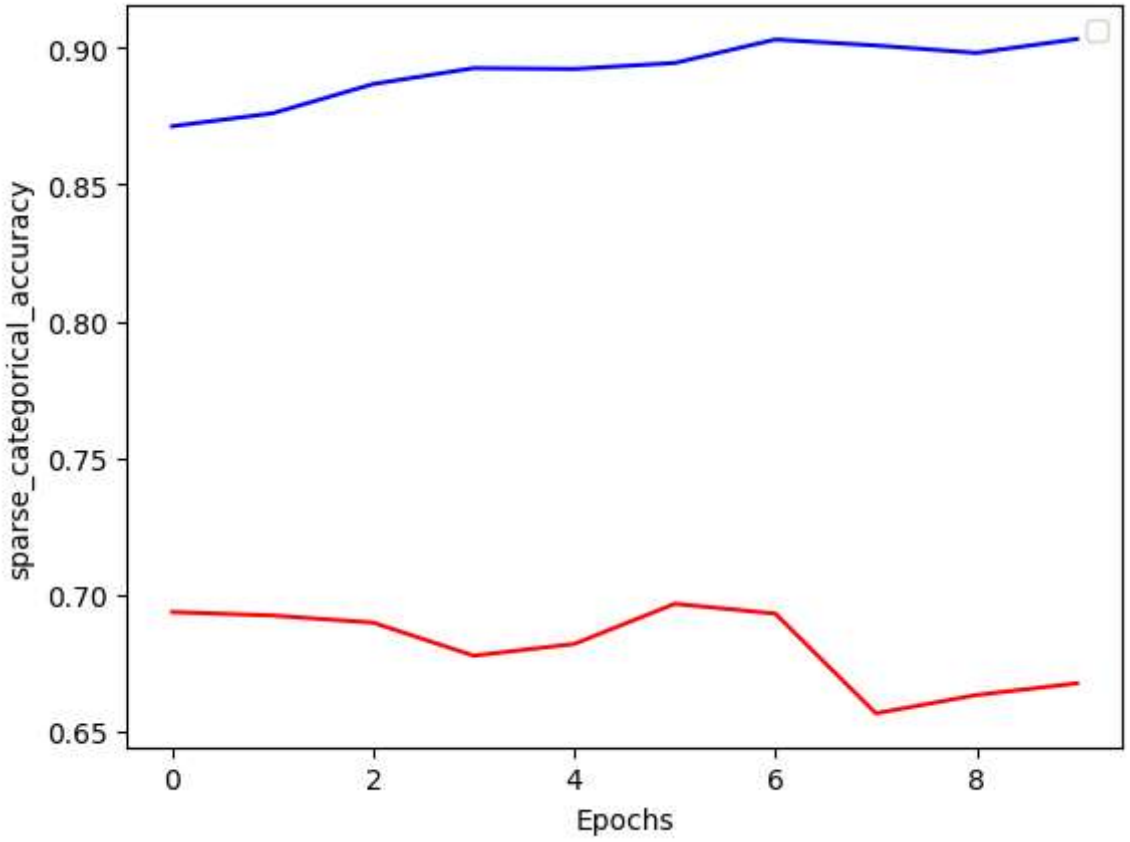
- Créez des batchs de 32 images pour votre jeu de données de validation et d'entrainement

In [29]:
```
ds_aug = ds_aug.batch(32)
ds_valid = ds_valid.batch(32)
```

In [30]:
```
plt.subplot()
plt.plot(history.history['accuracy'], color='blue')
plt.plot(history.history['val_accuracy'], color='red')
plt.xlabel('Epochs')
plt.ylabel('sparse_categorical_accuracy')
plt.legend()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note tha
t artists whose label start with an underscore are ignored when legend() is called
with no argument.
```

Out[30]:
```
<matplotlib.legend.Legend at 0x7fc5507f76d0>
```



- Réappliquez le même modèle sur votre nouveau jeu de données
  - N'oubliez pas de stocker votre entrainement dans une variable `history`

In [31]:
```
model = keras.models.Sequential()

model.add( keras.layers.Conv2D(16, (3,3),  activation='relu') )
model.add(keras.layers.BatchNormalization(axis=-1))
model.add( keras.layers.Conv2D(32, (3,3), activation='relu') )
model.add( keras.layers.MaxPooling2D((2,2)))
model.add( keras.layers.Conv2D(64, (3,3),  activation='relu') )
model.add(keras.layers.BatchNormalization(axis=-1))
model.add( keras.layers.Conv2D(32, (3,3), activation='relu') )
model.add( keras.layers.MaxPooling2D((2,2)))

model.add( keras.layers.Flatten())
model.add( keras.layers.Dense(512, activation='relu'))
model.add( keras.layers.Dense(10, activation='softmax'))

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [32]:
```
history = model.fit( ds_aug,
                     epochs          = 10,
                     verbose         = 1,
                     validation_data = ds_valid)
```

```
Epoch 1/10
15625/15625 [==============================] - 125s 8ms/step - loss: 2.7441 - accu
racy: 0.1001 - val_loss: 2.3094 - val_accuracy: 0.1001
Epoch 2/10
15625/15625 [==============================] - 125s 8ms/step - loss: 2.3158 - accu
racy: 0.0999 - val_loss: 3.0720 - val_accuracy: 0.0999
Epoch 3/10
15625/15625 [==============================] - 130s 8ms/step - loss: 2.3156 - accu
racy: 0.0998 - val_loss: 2.3188 - val_accuracy: 0.1001
Epoch 4/10
15625/15625 [==============================] - 126s 8ms/step - loss: 2.3153 - accu
racy: 0.0993 - val_loss: 2.3379 - val_accuracy: 0.1001
Epoch 5/10
15625/15625 [==============================] - 126s 8ms/step - loss: 2.3154 - accu
racy: 0.0997 - val_loss: 2.8197 - val_accuracy: 0.0999
Epoch 6/10
15625/15625 [==============================] - 124s 8ms/step - loss: 2.3155 - accu
racy: 0.1002 - val_loss: 10.1827 - val_accuracy: 0.0997
Epoch 7/10
15625/15625 [==============================] - 124s 8ms/step - loss: 2.3150 - accu
racy: 0.1002 - val_loss: 2.3153 - val_accuracy: 0.1000
Epoch 8/10
15625/15625 [==============================] - 124s 8ms/step - loss: 2.3151 - accu
racy: 0.1001 - val_loss: 2.6617 - val_accuracy: 0.0997
Epoch 9/10
15625/15625 [==============================] - 125s 8ms/step - loss: 2.3149 - accu
racy: 0.0993 - val_loss: 2.3102 - val_accuracy: 0.1000
Epoch 10/10
15625/15625 [==============================] - 123s 8ms/step - loss: 2.3148 - accu
racy: 0.0994 - val_loss: 3.4211 - val_accuracy: 0.1000
```

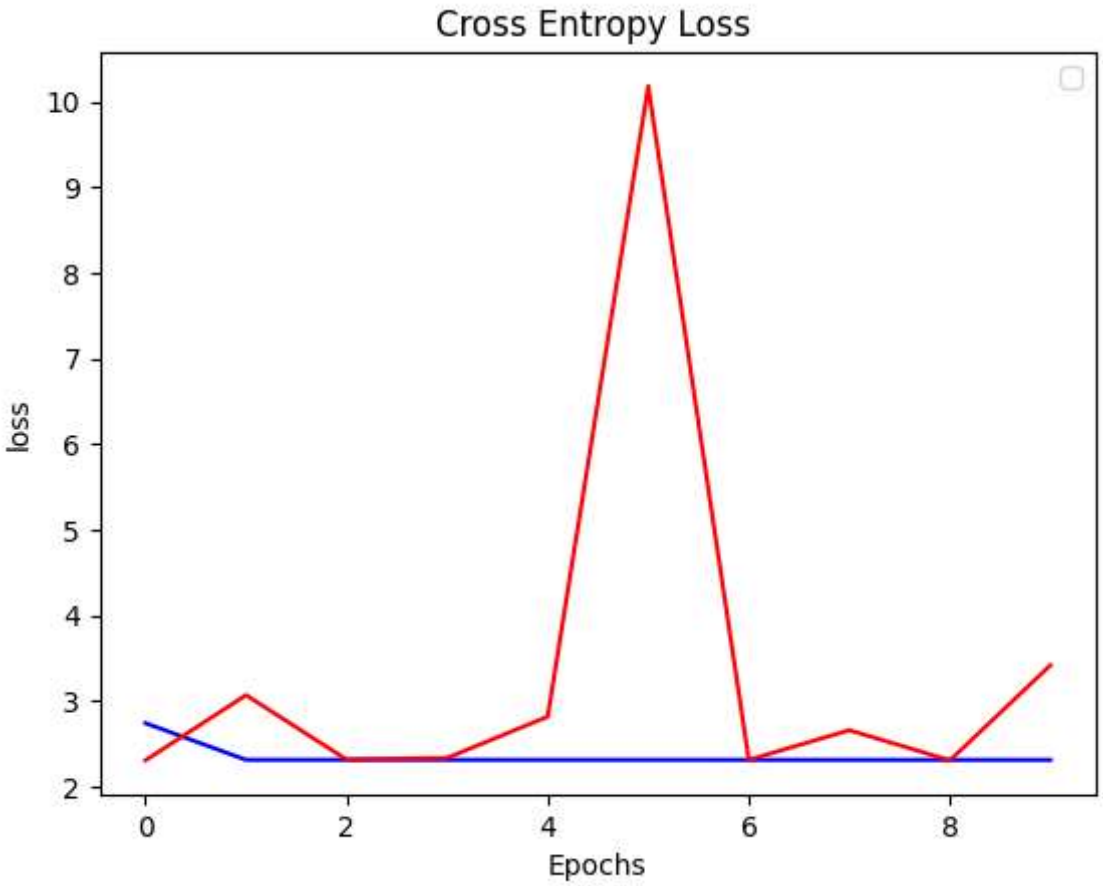
In [33]:

```
plt.subplot()
plt.title('Cross Entropy Loss ')
plt.plot(history.history['loss'], color='blue' )
plt.plot(history.history['val_loss'], color='red')
plt.xlabel('Epochs')
plt.ylabel('loss')
plt.legend()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note tha
t artists whose label start with an underscore are ignored when legend() is called
with no argument.
```

Out[33]:

```
<matplotlib.legend.Legend at 0x7fc5e4967250>
```



- Tentons une nouvelle technique de réguliarisation : le Dropout. Ajoutez quelques couches de dropout à 20% dans votre modèle

In [34]:
```
model = keras.models.Sequential()

model.add( keras.layers.Conv2D(16, (3,3),  activation='relu') )
model.add(keras.layers.BatchNormalization(axis=-1))
model.add( keras.layers.Conv2D(32, (3,3), activation='relu') )
model.add( keras.layers.MaxPooling2D((2,2)))
model.add(keras.layers.Dropout(0.2))
model.add( keras.layers.Conv2D(64, (3,3),  activation='relu') )
model.add(keras.layers.BatchNormalization(axis=-1))
model.add( keras.layers.Conv2D(32, (3,3), activation='relu') )
model.add( keras.layers.MaxPooling2D((2,2)))
model.add(keras.layers.Dropout(0.2))

model.add( keras.layers.Flatten())
model.add( keras.layers.Dense(512, activation='relu'))
model.add( keras.layers.Dense(10, activation='softmax'))

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

- Ré-entrainez votre modèle sur 15 à 30 epochs

In [ ]:
```
history = model.fit( ds_aug,
                     epochs          = 15,
                     verbose         = 1,
                     validation_data = ds_valid)
```

```
Epoch 1/15
15625/15625 [==============================] - 106s 7ms/step - loss: 2.6619 - accu
racy: 0.1000 - val_loss: 2.3124 - val_accuracy: 0.1000
Epoch 2/15
15625/15625 [==============================] - 105s 7ms/step - loss: 2.3155 - accu
racy: 0.0996 - val_loss: 2.3182 - val_accuracy: 0.1000
Epoch 3/15
15625/15625 [==============================] - 105s 7ms/step - loss: 2.3154 - accu
racy: 0.1002 - val_loss: 2.3162 - val_accuracy: 0.1000
Epoch 4/15
15625/15625 [==============================] - 105s 7ms/step - loss: 2.3156 - accu
racy: 0.1000 - val_loss: 2.3140 - val_accuracy: 0.1000
Epoch 5/15
15625/15625 [==============================] - 104s 7ms/step - loss: 2.3154 - accu
racy: 0.1001 - val_loss: 2.3167 - val_accuracy: 0.1000
Epoch 6/15
15625/15625 [==============================] - 104s 7ms/step - loss: 2.3155 - accu
racy: 0.0998 - val_loss: 2.3084 - val_accuracy: 0.1000
Epoch 7/15
15625/15625 [==============================] - 105s 7ms/step - loss: 2.3151 - accu
racy: 0.0999 - val_loss: 2.3130 - val_accuracy: 0.1000
Epoch 8/15
15625/15625 [==============================] - 103s 7ms/step - loss: 2.3150 - accu
racy: 0.0994 - val_loss: 2.3180 - val_accuracy: 0.1000
Epoch 9/15
15625/15625 [==============================] - 105s 7ms/step - loss: 2.3151 - accu
racy: 0.1004 - val_loss: 2.3051 - val_accuracy: 0.1000
Epoch 10/15
15625/15625 [==============================] - 104s 7ms/step - loss: 2.3150 - accu
racy: 0.0993 - val_loss: 2.3075 - val_accuracy: 0.1000
Epoch 11/15
15625/15625 [==============================] - 104s 7ms/step - loss: 2.3151 - accu
racy: 0.1000 - val_loss: 2.3143 - val_accuracy: 0.1000
Epoch 12/15
15625/15625 [==============================] - 104s 7ms/step - loss: 2.3150 - accu
racy: 0.1002 - val_loss: 2.3434 - val_accuracy: 0.1000
Epoch 13/15
15625/15625 [==============================] - 104s 7ms/step - loss: 2.3149 - accu
racy: 0.0999 - val_loss: 2.3155 - val_accuracy: 0.1000
Epoch 14/15
15625/15625 [==============================] - 106s 7ms/step - loss: 2.3146 - accu
racy: 0.0993 - val_loss: 2.3091 - val_accuracy: 0.1000
Epoch 15/15
15625/15625 [==============================] - 106s 7ms/step - loss: 2.3144 - accu
racy: 0.1002 - val_loss: 2.3195 - val_accuracy: 0.1000
```

- Faites les visualisations de de loss et d'accuracy

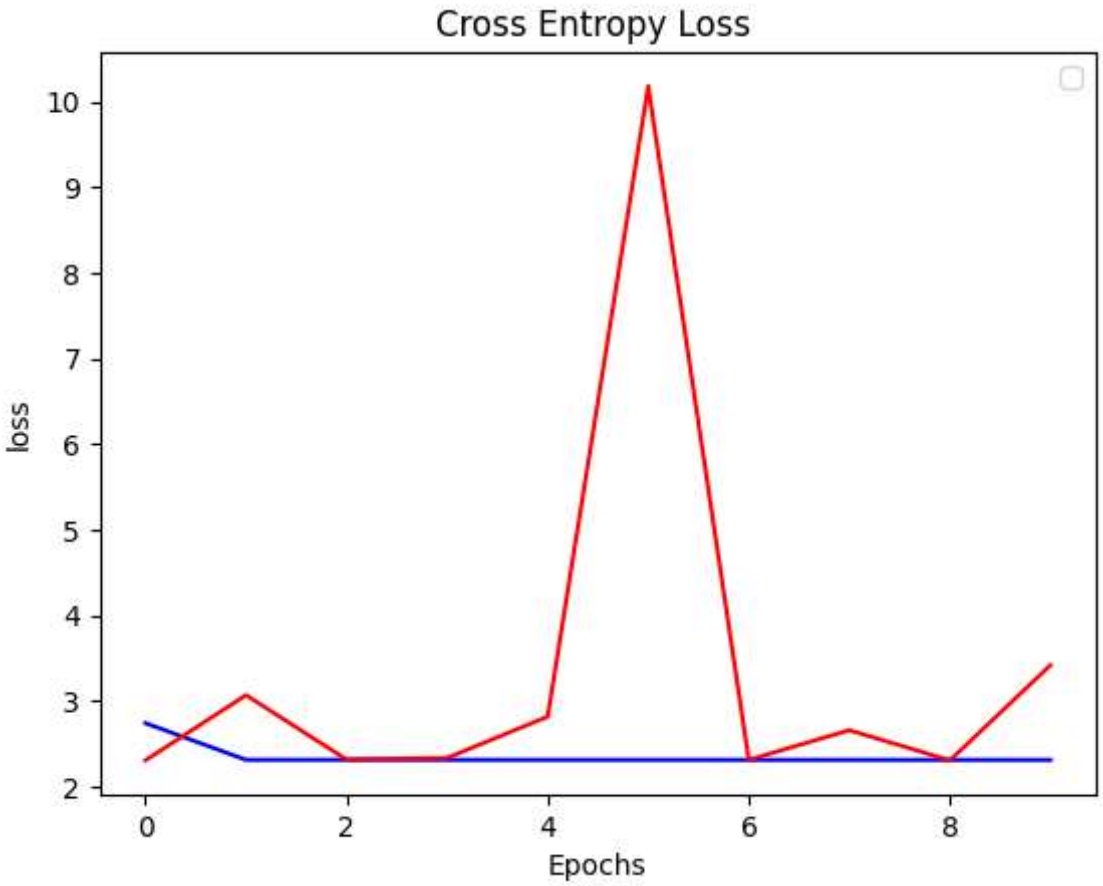
In [36]:

```
plt.subplot()
plt.title('Cross Entropy Loss ')
plt.plot(history.history['loss'], color='blue' )
plt.plot(history.history['val_loss'], color='red')
plt.xlabel('Epochs')
plt.ylabel('loss')
plt.legend()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note tha
t artists whose label start with an underscore are ignored when legend() is called
with no argument.
```

Out[36]:

```
<matplotlib.legend.Legend at 0x7fc67317eda0>
```



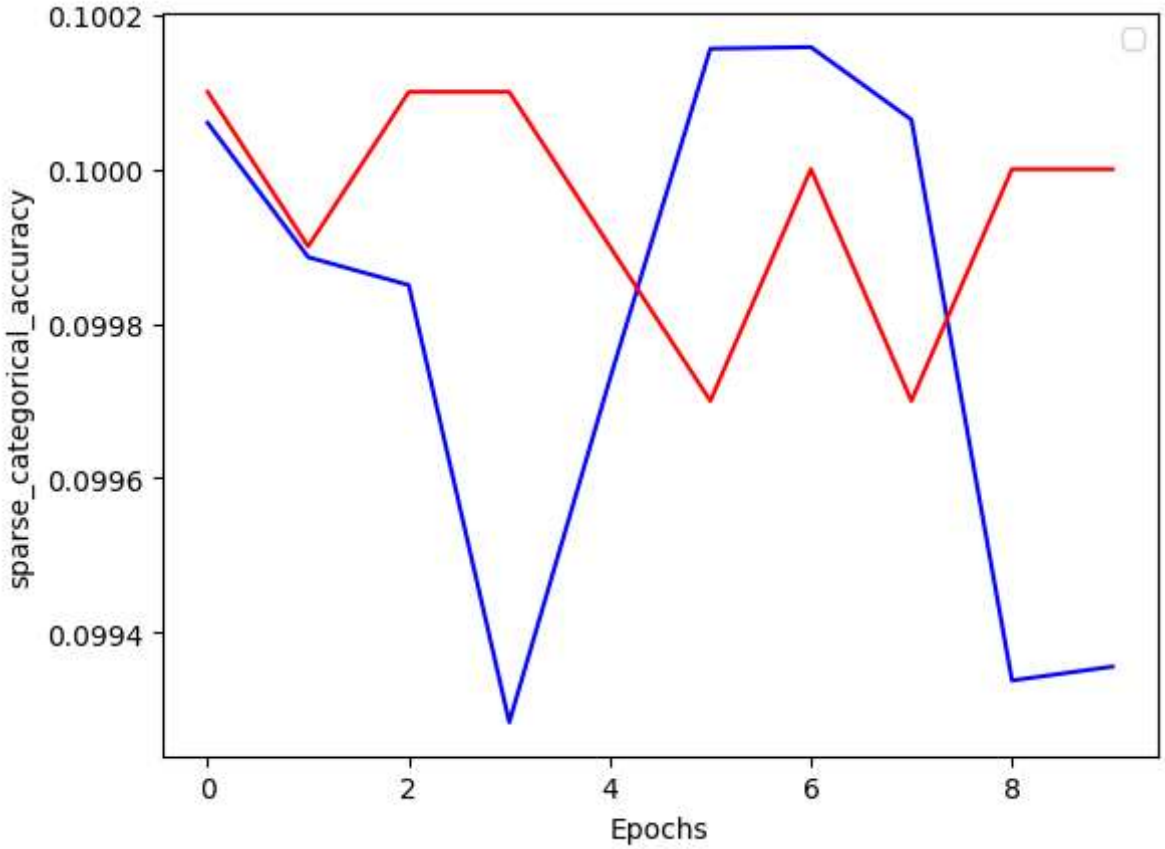
In [35]:

```
plt.subplot()
plt.plot(history.history['accuracy'], color='blue')
plt.plot(history.history['val_accuracy'], color='red')
plt.xlabel('Epochs')
plt.ylabel('sparse_categorical_accuracy')
plt.legend()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note tha
t artists whose label start with an underscore are ignored when legend() is called
with no argument.
```

Out[35]:

```
<matplotlib.legend.Legend at 0x7fc5edcfb220>
```



---> Pas mal, on arrive à des scores tout à fait honorables !

In [ ]: