Homework2DenseNet

February 28, 2022

1 Homework 2 DenseNet121

1.1 Loading Libraries

[1]:
```
# Importing Libraries



# Basic Libraries␣
 ↪#############################################################################
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import seaborn as sns
import datetime
import time


# For Feature Engineering␣
 ↪#############################################################################


# For Machine Learning Techniques␣
 ↪#############################################################
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import cifar10

from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications.densenet import preprocess_input


# For Data Anaylsis␣
 ↪#############################################################################
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import classification_report, confusion_matrix


# Personal Preference␣
↪##########################################################################
import warnings
warnings.filterwarnings('ignore')
```

from tensorflow.keras.preprocessing import image from tensorflow.keras.preprocessing.image import ImageDataGenerator,img_to_array

from tensorflow.keras.models import Model from tensorflow.keras.optimizers import Adam from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau

from tensorflow.keras.layers import Dense,GlobalAveragePooling2D,Convolution2D,BatchNormalization from tensorflow.keras.layers import Flatten,MaxPooling2D,Dropout

1.1.1 Setting GPU

[2]:
```
# Change to markdown if gpu is not set up

import os

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())

physical_devices = tf.config.list_physical_devices("GPU")
tf.config.experimental.set_memory_growth(physical_devices[0], True)
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 13338002541111118179
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 6273040384
locality {
  bus_id: 1
  links {
  }
}
incarnation: 5586588163853313758
physical_device_desc: "device: 0, name: NVIDIA GeForce RTX 2070 SUPER, pci bus
id: 0000:02:00.0, compute capability: 7.5"
```

```
xla_global_id: 416903419
]
```

1.2 Loading Data

[3]:
```
# Setting Class Names

class_names=['airplane','automobile','bird','cat','deer',
             'dog','frog','horse','ship','truck']
```

[4]:
```
# Loading the Dataset

(x_train,y_train),(x_test,y_test)=cifar10.load_data()
```

[5]:
```
# Normalizing the Images

x_train=x_train/255.0
print(x_train.shape)

x_test=x_test/255.0
print(x_test.shape)
```

```
(50000, 32, 32, 3)
(10000, 32, 32, 3)
```

1.3 Splitting the Data

[6]:
```
# 10% of the Orginal Dataset

x_train10, not_needed1, y_train10, not_needed2 = train_test_split(
    x_train, y_train, test_size=0.90, random_state=42)
```

[7]:
```
# 50% of the Orginal Dataset

x_train50, not_needed1, y_train50, not_needed2 = train_test_split(
    x_train, y_train, test_size=0.50, random_state=42)
```

[8]:
```
# 80% of the Orginal Dataset (redundant)

x_train80, not_needed1, y_train80, not_needed2 = train_test_split(
    x_train, y_train, test_size=0.20, random_state=42)
```

1.4 Setting up Models

[9]:
```
# Creating the actual model

model_10 = tf.keras.applications.DenseNet121(
    include_top=True,
    weights=None,
    input_tensor=None,
    input_shape=(32, 32, 3),
    pooling=None,
    classes=1000,)
```

[10]:
```
# Setting up Mutiple Models

model_50 = model_10

model_80 = model_10
```

[11]:
```
# Model Compiling

model_10.compile(loss="sparse_categorical_crossentropy",
                      optimizer="Adam", metrics=["sparse_categorical_accuracy"])

model_50.compile(loss="sparse_categorical_crossentropy",
                      optimizer="Adam", metrics=["sparse_categorical_accuracy"])

model_80.compile(loss="sparse_categorical_crossentropy",
                      optimizer="Adam", metrics=["sparse_categorical_accuracy"])
```

1.5 Training the Models

[12]:
```
# Fitting the 10% Model

history10 = model_10.fit(x_train10,y_train10,epochs=10)
```

```
Epoch 1/10
157/157 [==============================] - 25s 71ms/step - loss: 2.1026 -
sparse_categorical_accuracy: 0.2984
Epoch 2/10
157/157 [==============================] - 10s 62ms/step - loss: 1.7068 -
sparse_categorical_accuracy: 0.3742
Epoch 3/10
157/157 [==============================] - 10s 64ms/step - loss: 1.4992 -
sparse_categorical_accuracy: 0.4578
Epoch 4/10
157/157 [==============================] - 10s 62ms/step - loss: 1.4213 -
sparse_categorical_accuracy: 0.4838
```

```
Epoch 5/10
157/157 [==============================] - 10s 62ms/step - loss: 1.2623 -
sparse_categorical_accuracy: 0.5432
Epoch 6/10
157/157 [==============================] - 10s 62ms/step - loss: 1.1838 -
sparse_categorical_accuracy: 0.5740
Epoch 7/10
157/157 [==============================] - 10s 65ms/step - loss: 1.0485 -
sparse_categorical_accuracy: 0.6296
Epoch 8/10
157/157 [==============================] - 10s 63ms/step - loss: 0.9555 -
sparse_categorical_accuracy: 0.6622
Epoch 9/10
157/157 [==============================] - 10s 61ms/step - loss: 0.8585 -
sparse_categorical_accuracy: 0.6952
Epoch 10/10
157/157 [==============================] - 10s 61ms/step - loss: 0.7603 -
sparse_categorical_accuracy: 0.7364
```

[13]:
```
# Fitting the 50% Model

history50 = model_50.fit(x_train50,y_train50,epochs=10)
```

```
Epoch 1/10
782/782 [==============================] - 49s 63ms/step - loss: 1.1217 -
sparse_categorical_accuracy: 0.6128
Epoch 2/10
782/782 [==============================] - 48s 61ms/step - loss: 0.9520 -
sparse_categorical_accuracy: 0.6694
Epoch 3/10
782/782 [==============================] - 49s 62ms/step - loss: 0.8130 -
sparse_categorical_accuracy: 0.7200
Epoch 4/10
782/782 [==============================] - 49s 62ms/step - loss: 0.7176 -
sparse_categorical_accuracy: 0.7540
Epoch 5/10
782/782 [==============================] - 48s 62ms/step - loss: 0.6150 -
sparse_categorical_accuracy: 0.7871
Epoch 6/10
782/782 [==============================] - 48s 62ms/step - loss: 0.5415 -
sparse_categorical_accuracy: 0.8129
Epoch 7/10
782/782 [==============================] - 48s 61ms/step - loss: 0.4448 -
sparse_categorical_accuracy: 0.8457
Epoch 8/10
782/782 [==============================] - 49s 62ms/step - loss: 0.3821 -
sparse_categorical_accuracy: 0.8667
Epoch 9/10
```

```
782/782 [==============================] - 49s 63ms/step - loss: 0.3162 -
sparse_categorical_accuracy: 0.8884
Epoch 10/10
782/782 [==============================] - 49s 62ms/step - loss: 0.2573 -
sparse_categorical_accuracy: 0.9107
```

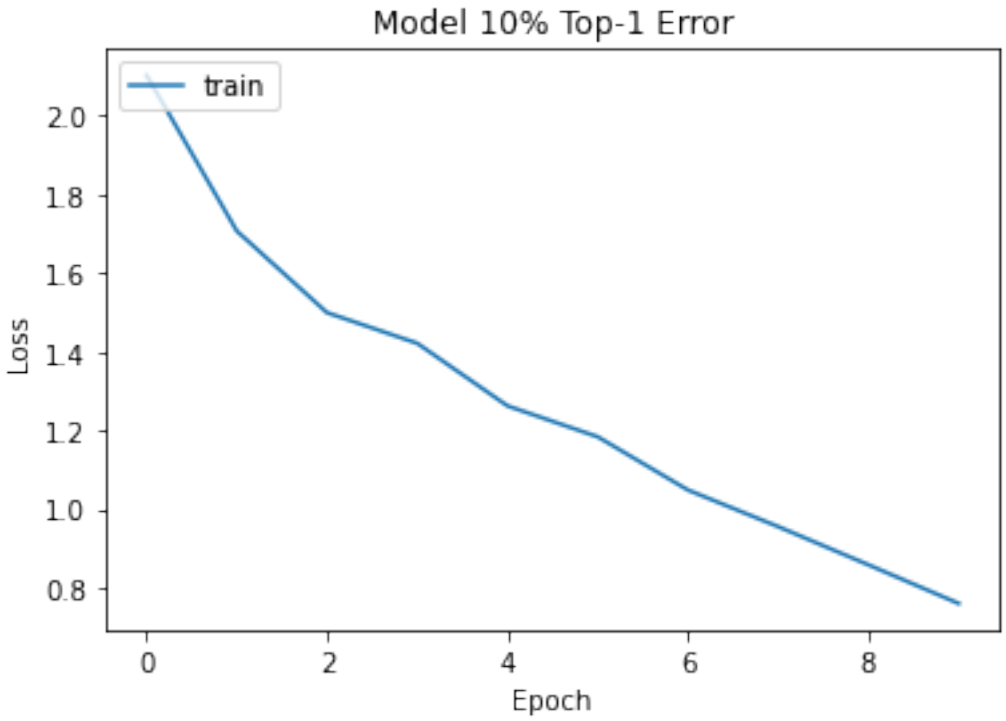
[14]:
```
# Fitting the 80% Model

history80 = model_80.fit(x_train80,y_train80,epochs=10)
```

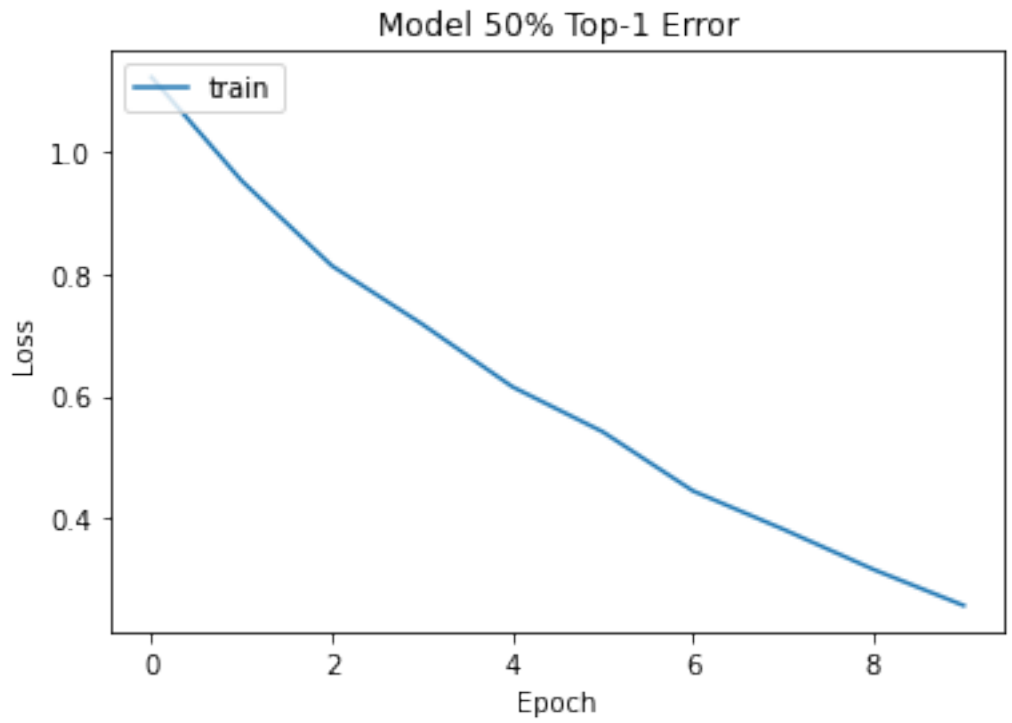
```
Epoch 1/10
1250/1250 [==============================] - 84s 64ms/step - loss: 0.4941 -
sparse_categorical_accuracy: 0.8419
Epoch 2/10
1250/1250 [==============================] - 79s 63ms/step - loss: 0.3693 -
sparse_categorical_accuracy: 0.8741
Epoch 3/10
1250/1250 [==============================] - 79s 63ms/step - loss: 0.2946 -
sparse_categorical_accuracy: 0.8997
Epoch 4/10
1250/1250 [==============================] - 80s 64ms/step - loss: 0.2465 -
sparse_categorical_accuracy: 0.9155
Epoch 5/10
1250/1250 [==============================] - 80s 64ms/step - loss: 0.2034 -
sparse_categorical_accuracy: 0.9284
Epoch 6/10
1250/1250 [==============================] - 78s 62ms/step - loss: 0.1748 -
sparse_categorical_accuracy: 0.9396
Epoch 7/10
1250/1250 [==============================] - 78s 62ms/step - loss: 0.1537 -
sparse_categorical_accuracy: 0.9460
Epoch 8/10
1250/1250 [==============================] - 78s 63ms/step - loss: 0.1345 -
sparse_categorical_accuracy: 0.9531
Epoch 9/10
1250/1250 [==============================] - 78s 63ms/step - loss: 0.1236 -
sparse_categorical_accuracy: 0.9574
Epoch 10/10
1250/1250 [==============================] - 79s 63ms/step - loss: 0.1132 -
sparse_categorical_accuracy: 0.9594
```

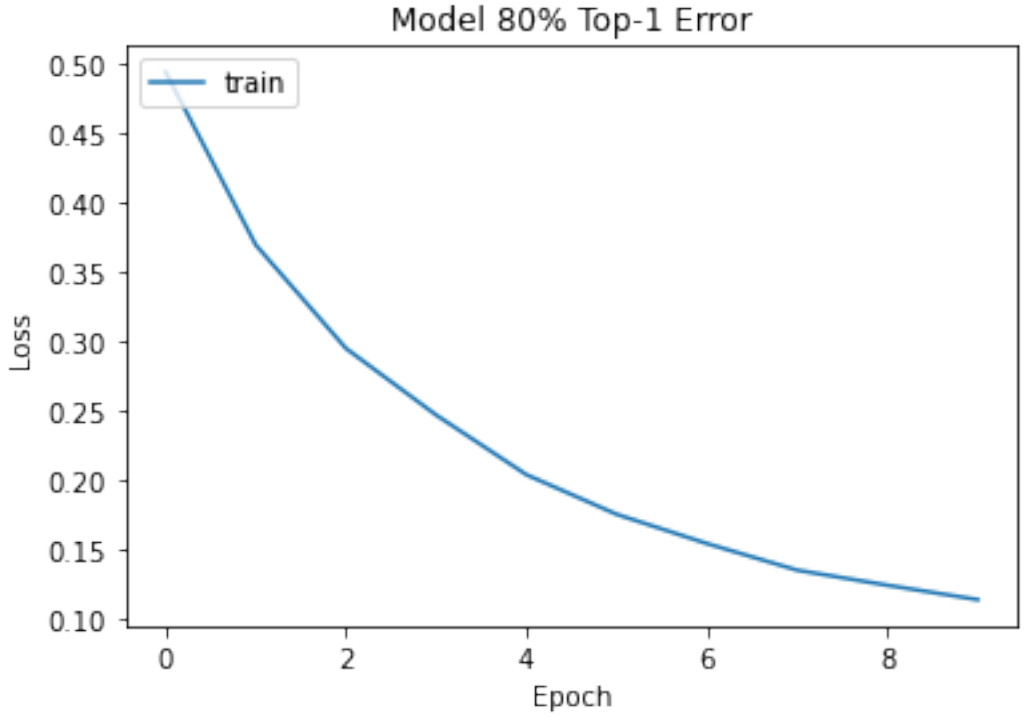
1.6 Plotting the Models

[15]:
```
# Plotting the 10% Model

plt.plot(history10.history['loss'])
plt.title('Model 10% Top-1 Error')
```

```
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train'], loc = 'upper left')
plt.show()

# Plotting the 50% Model

plt.plot(history50.history['loss'])
plt.title('Model 50% Top-1 Error')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train'], loc = 'upper left')
plt.show()

# Plotting the 80% Model

plt.plot(history80.history['loss'])
plt.title('Model 80% Top-1 Error')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train'], loc = 'upper left')
plt.show()
```




Model 50% Top-1 Error
train
1.0
0.8
0.6
0.4
Loss
0
2
4
6
8
Epoch


Model 80% Top-1 Error
train
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
Loss
0
2
4
6
8
Epoch

2 Homework 2 My Model

2.1 Setting up the Model

[16]:
```
# Designing the Model


cifar10_model=tf.keras.models.Sequential()

# Convultions␣
↪#################################################################################

# First Layer
cifar10_model.add(tf.keras.layers.
↪Conv2D(filters=32,kernel_size=3,padding="same",
                                        activation="relu",␣
↪input_shape=[32,32,3]))

# Max Pooling Layer
cifar10_model.add(tf.keras.layers.
↪MaxPool2D(pool_size=2,strides=2,padding='valid'))

# Third Layer
cifar10_model.add(tf.keras.layers.
↪Conv2D(filters=64,kernel_size=3,padding="same",
                                        activation="relu"))

# Max Pooling Layer
cifar10_model.add(tf.keras.layers.
↪MaxPool2D(pool_size=2,strides=2,padding='valid'))

# Flattening Layer
cifar10_model.add(tf.keras.layers.Flatten())

# Droput Layer
cifar10_model.add(tf.keras.layers.Dropout(0.5,noise_shape=None,seed=None))


# Neural Network␣
↪#################################################################################

# Adding the first fully connected layer
cifar10_model.add(tf.keras.layers.Dense(units=128,activation='relu'))

# Output Layer
cifar10_model.add(tf.keras.layers.Dense(units=10,activation='softmax'))
```

```
cifar10_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 16, 16, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 flatten (Flatten)           (None, 4096)              0

 dropout (Dropout)           (None, 4096)              0

 dense (Dense)               (None, 128)               524416

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 545,098
Trainable params: 545,098
Non-trainable params: 0
_________________________________________________________________
```

[17]:
```
# Setting up the Different Versions

cifar10_model_10 = cifar10_model
cifar10_model_50 = cifar10_model
cifar10_model_80 = cifar10_model
```

[18]:
```
# Compiling Models

cifar10_model_10.compile(loss="sparse_categorical_crossentropy",
                         optimizer="Adam", metrics=["sparse_categorical_accuracy"])

cifar10_model_50.compile(loss="sparse_categorical_crossentropy",
                         optimizer="Adam", metrics=["sparse_categorical_accuracy"])

cifar10_model_80.compile(loss="sparse_categorical_crossentropy",
                         optimizer="Adam", metrics=["sparse_categorical_accuracy"])
```

2.2 Training My Models

[19]:
```
# Train 10% Model

history10_2 = cifar10_model_10.fit(x_train10,y_train10,epochs=10)
```

```
Epoch 1/10
157/157 [==============================] - 1s 4ms/step - loss: 2.0136 -
sparse_categorical_accuracy: 0.2524
Epoch 2/10
157/157 [==============================] - 1s 3ms/step - loss: 1.6405 -
sparse_categorical_accuracy: 0.4162
Epoch 3/10
157/157 [==============================] - 1s 3ms/step - loss: 1.4794 -
sparse_categorical_accuracy: 0.4670
Epoch 4/10
157/157 [==============================] - 1s 4ms/step - loss: 1.3454 -
sparse_categorical_accuracy: 0.5200
Epoch 5/10
157/157 [==============================] - 1s 4ms/step - loss: 1.2822 -
sparse_categorical_accuracy: 0.5402
Epoch 6/10
157/157 [==============================] - 1s 4ms/step - loss: 1.1811 -
sparse_categorical_accuracy: 0.5712
Epoch 7/10
157/157 [==============================] - 1s 4ms/step - loss: 1.1186 -
sparse_categorical_accuracy: 0.6024
Epoch 8/10
157/157 [==============================] - 1s 3ms/step - loss: 1.0437 -
sparse_categorical_accuracy: 0.6332
Epoch 9/10
157/157 [==============================] - 1s 4ms/step - loss: 0.9807 -
sparse_categorical_accuracy: 0.6584
Epoch 10/10
157/157 [==============================] - 1s 4ms/step - loss: 0.9361 -
sparse_categorical_accuracy: 0.6572
```

[20]:
```
# Train 50% Model

history50_2 = cifar10_model_50.fit(x_train50,y_train50,epochs=10)
```

```
Epoch 1/10
782/782 [==============================] - 3s 3ms/step - loss: 1.1954 -
sparse_categorical_accuracy: 0.5849
Epoch 2/10
782/782 [==============================] - 3s 3ms/step - loss: 1.0532 -
sparse_categorical_accuracy: 0.6284
Epoch 3/10
```

```
782/782 [==============================] - 3s 3ms/step - loss: 0.9669 -
sparse_categorical_accuracy: 0.6566
Epoch 4/10
782/782 [==============================] - 3s 4ms/step - loss: 0.8949 -
sparse_categorical_accuracy: 0.6828
Epoch 5/10
782/782 [==============================] - 3s 4ms/step - loss: 0.8353 -
sparse_categorical_accuracy: 0.7051
Epoch 6/10
782/782 [==============================] - 3s 4ms/step - loss: 0.7844 -
sparse_categorical_accuracy: 0.7214
Epoch 7/10
782/782 [==============================] - 3s 4ms/step - loss: 0.7414 -
sparse_categorical_accuracy: 0.7378
Epoch 8/10
782/782 [==============================] - 3s 4ms/step - loss: 0.6896 -
sparse_categorical_accuracy: 0.7550
Epoch 9/10
782/782 [==============================] - 3s 4ms/step - loss: 0.6650 -
sparse_categorical_accuracy: 0.7627
Epoch 10/10
782/782 [==============================] - 3s 4ms/step - loss: 0.6135 -
sparse_categorical_accuracy: 0.7834
```

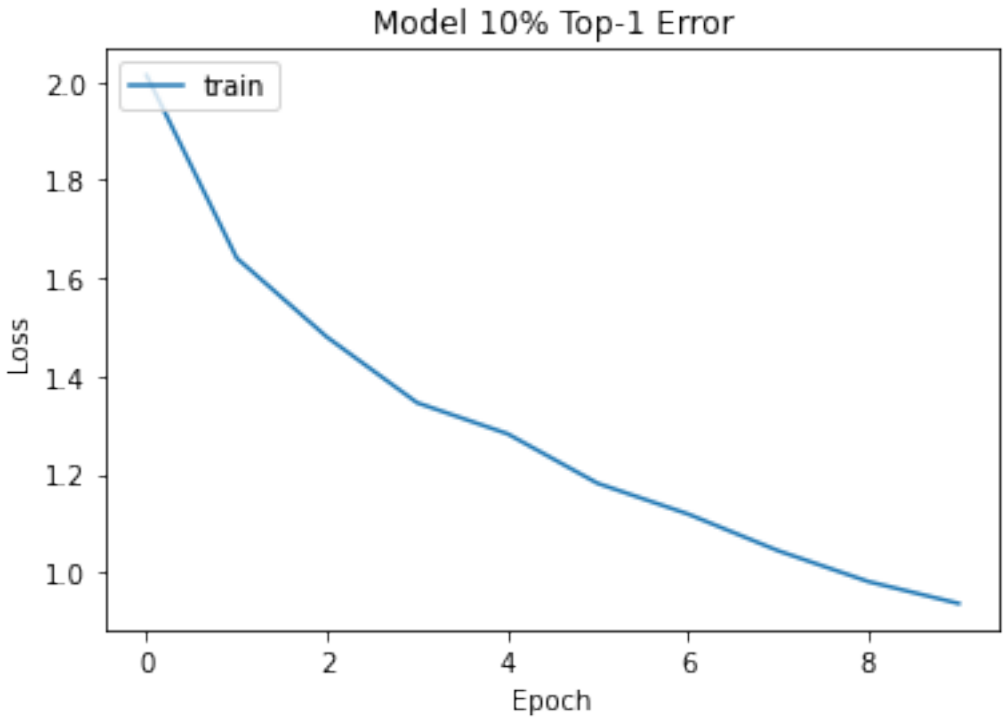
[21]:
```
# Train 10% Model

history80_2 = cifar10_model_80.fit(x_train80,y_train80,epochs=10)
```

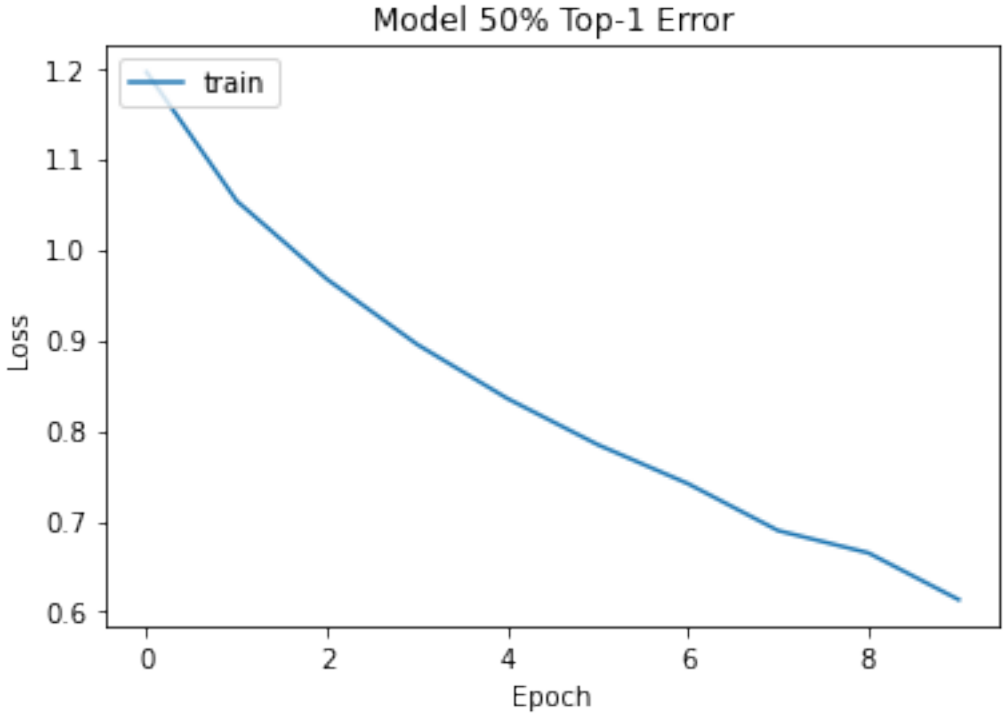
```
Epoch 1/10
1250/1250 [==============================] - 5s 4ms/step - loss: 0.7775 -
sparse_categorical_accuracy: 0.7328
Epoch 2/10
1250/1250 [==============================] - 5s 4ms/step - loss: 0.7122 -
sparse_categorical_accuracy: 0.7521
Epoch 3/10
1250/1250 [==============================] - 5s 4ms/step - loss: 0.6692 -
sparse_categorical_accuracy: 0.7646
Epoch 4/10
1250/1250 [==============================] - 5s 4ms/step - loss: 0.6280 -
sparse_categorical_accuracy: 0.7780
Epoch 5/10
1250/1250 [==============================] - 5s 4ms/step - loss: 0.5989 -
sparse_categorical_accuracy: 0.7871
Epoch 6/10
1250/1250 [==============================] - 5s 4ms/step - loss: 0.5764 -
sparse_categorical_accuracy: 0.7969
Epoch 7/10
1250/1250 [==============================] - 5s 4ms/step - loss: 0.5442 -
```

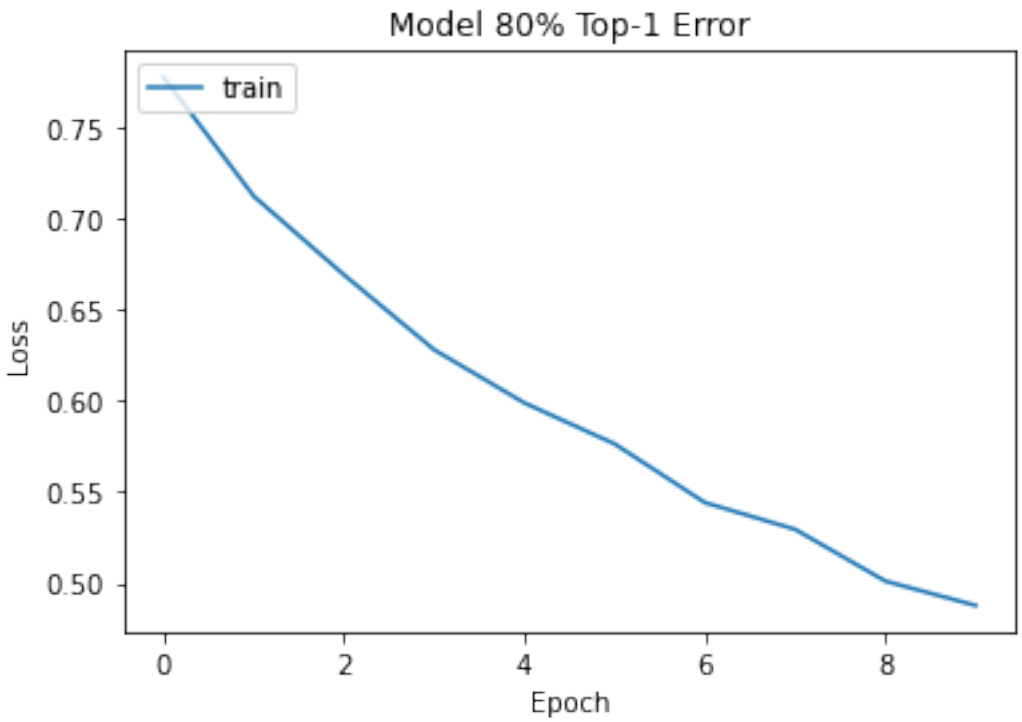
```
sparse_categorical_accuracy: 0.8055
Epoch 8/10
1250/1250 [==============================] - 5s 4ms/step - loss: 0.5296 -
sparse_categorical_accuracy: 0.8138
Epoch 9/10
1250/1250 [==============================] - 5s 4ms/step - loss: 0.5012 -
sparse_categorical_accuracy: 0.8209
Epoch 10/10
1250/1250 [==============================] - 5s 4ms/step - loss: 0.4878 -
sparse_categorical_accuracy: 0.8260
```

2.3 Plotting the Loss of My Model

[22]:
```
# Plotting the 10% Model

plt.plot(history10_2.history['loss'])
plt.title('Model 10% Top-1 Error')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train'], loc = 'upper left')
plt.show()

# Plotting the 50% Model

plt.plot(history50_2.history['loss'])
plt.title('Model 50% Top-1 Error')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train'], loc = 'upper left')
plt.show()

# Plotting the 80% Model

plt.plot(history80_2.history['loss'])
plt.title('Model 80% Top-1 Error')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train'], loc = 'upper left')
plt.show()
```


Model 10% Top-1 Error
train
Loss
Epoch
2.0
1.8
1.6
1.4
1.2
1.0
0
2
4
6
8


Model 50% Top-1 Error
train
Loss
Epoch
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0
2
4
6
8


Model 80% Top-1 Error
train
0.75
0.70
0.65
0.60
0.55
0.50
Loss
0
2
4
6
8
Epoch