Preston Robertson

IE 8990
Spring 2022
Homework #2
Due Date: 3/8/2022 5PM CST
Submission: Please put your answer and code in a PDF file and upload on Canvas

---

Q1. Consider a 1-dimensional time-series with values 2, 1, 3, 4, 7. Perform a convolution with a 1-dimensional filter (1, 0, 1) and zero padding with stride of 1.

- The answer is (3,4,7)
  - The pattern is noticeable, that the farther numbers on the right are higher so as the stride moves along this 1D set of values the stride will pick up the right most value. Which is plausible because the final filter number is 1.

Q2. Consider an activation volume of size 13 × 13 × 64 and a filter of size 3 × 3 × 64. Discuss whether it is possible to perform convolutions with strides 2, 3, 4, and justify your answer in each case.

- Output Size = ((Input Size + (2 * Padding Size) – Filter Size)/stride) + 1
  - Using this equation, we can see the resulting matrix to verify if it is possible.
- In the next table I used the above matrix to see if the results are a whole number, if it is yes then it is possible to perform convulsions
  - In some instances, it is important to change the padding size to allow for the convolutional layer to be used.

Table 1: Answer to Question 1

Stride	With No Padding	Verification	With Padding of 2	Verification
2	6	Yes	N/A	N/A
3	4.333333	No	5	Yes
4	3.5	No	4	Yes
5	3	Yes	N/A	N/A

Q3. Based on the input image in Table 1:

Table 1: Input

6	3	4	4	5	0	3
4	7	4	0	4	0	4
7	0	3	4	4	5	2
3	7	0	3	5	0	7
5	8	2	5	5	4	2
8	0	0	6	6	0	0
6	4	3	0	0	4	5

Table 2: Horizontal edge detection filter

1	1	1
0	0	0
-1	-1	-1

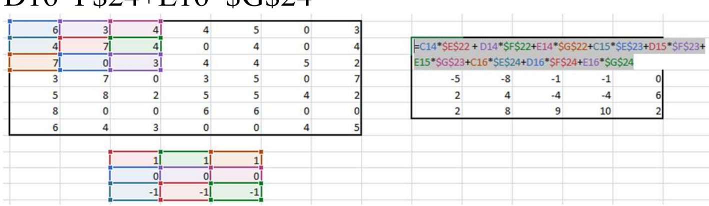
- Computer the convolution of the input volume with the horizontal edge detection filter in Table 2. Use stride of 1 without padding
  - Output Size = ((Input Size + (2 * Padding Size) – Filter Size)/stride) + 1
    - Using this equation, we can see that a 5x5 matrix is the result.
- I used Excel to solve this problem, the equation is here:
  - C14*E22 +D14*F22+E14*G22+C15*E23+D15*F23+E15*G23+C16*E24+D16*F$24+E16*$G$24
  - 

Table 3: Question 3 Part 1

3	4	2	-4	-3
5	1	0	-4	-4
-5	-8	-1	-1	0
2	4	-4	-4	6
2	8	9	10	2

- Perform a 4×4 pooling at stride 1 on the result from the previous question
    - I used the max pooling technique and got the results below.
    - Output Size = ((Input Size + (2 * Padding Size) – Filter Size)/stride) + 1
        - Using this equation, we can see that a 2x2 matrix is the result.

Table 4: Question 3 Part 2

5	6
10	10

Q4. Randomly select 20% of the CIFAR-10 data as test data. Train the network on subsets of varying size (10%, 50%, 80%) from the remaining CIFAR-10 data. Perform the following tasks:

- Train DenseNet121 architecture. Plot the top-1 error with data size.
- Design a CNN model based on your own choice. Plot the top-1 error with data size.
    - At the bottom of the document.

Q5. Review: Select one article from below and summarize it (1 page, single-space, font size 12pt)

- Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., Kolter, Z. (2019). Differentiable convex optimization layers. arXiv preprint arXiv:1910.12430.

This article is about how Agrawal et al. applied the principles of Domain-Specific Languages (DSLs) to differentiable convex optimization problems. The DSL allowed for non-experts in optimization to have access to convex optimization. One way this paper accomplishes this is through making a standard language for differentiable convex optimization, they call the grammar “disciplined parametrized programming”. One intention of this standardized grammar is to allow for use of differentiable convex optimization in standardized programs such as PyTorch and TensorFlow 2.0. This lowers the barrier entry significantly. To clarify, an optimization problem is like problems from linear programming. It is minimizing or maximizing the problem with some constraint. In practice, this way of solving problems can mathematically solve problems far faster than other methods. This is due to multi-threading on CPU and GPU. The convex optimization problem is one of the solutions to our new activation functions tunable parameter. Back to the summary, another novel point from the papers is the use of abstract forms instead of conic forms. This has the advantage of being able to quickly experiment through new problems that did not have conic forms. This method still allows for conic form problems to be solved. Using abstract forms means the user does not need to pre-process data in conic forms allowing for more ease of use for non-experts. This abstract form allows for dynamic learning. This dynamic learning comes from the ability to minimize across the entire set of options. Whereas stochastic gradient descent in neural networks is short sighted based on mini-batch size. The convex optimization problem can understand all options and pick the most optimal choice given the problem domain. This is one of the reasons why optimization problems have such popularity as a modeling tool.

One of the main difficulties when using optimization for a problem is tracking change. In stochastic gradient descent, how you optimize your problem is by showing how the changes in each weight effects the loss. This change is derivation to show the impact/importance of a specific weight. Optimization lacked the ability to show the change in the solution based on certain parameters since it changes for each problem. This paper combats this by showing a standardized way to take the derivative of any differentiable convex optimization problem to map the importance of specific inputs. This allows for the optimization problem to be solved much more quickly than before, making it a viable solution. Since the entirety of neural network training is essentially an optimization problem, this method is now a potential solution to training speed and accuracy. The use of convex optimization will allow for quicker grid search commands for non-neural network based machine learning techniques as well. How are the values retrieved from the solver? The solver finds the optimal input that would satisfy the conic form of the problem then reshaping and scaling to the original problem.

Overall, this paper sought to explore the use of non-conic optimization problems in machine learning. This would significantly reduce training time if ever implemented; however, the optimization problems were too complex for non-experts, so the paper made its own grammar that would work with TensorFlow and PyTorch. This gives the ability to use these optimization problems to non-experts. Finally, this paper found a way through the use of derivates that it is possible to solve non-conic optimization problems. This will have large impact when grid-searching hyper parameters for several hyperparameters.

Homework2DenseNet

March 8, 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: 9327267842447527460
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 6273040384
locality {
  bus_id: 1
  links {
  }
}
incarnation: 6856064255926636361
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

model_10.fit(x_train10,y_train10,epochs=25, batch_size=32, validation_split=0.2)
#history10 = model_10.fit(x_train10,y_train10,epochs=25, batch_size=32,␣
 ↪validation_split=0.2)
```

```
Epoch 1/25
125/125 [==============================] - 22s 86ms/step - loss: 2.1706 -
sparse_categorical_accuracy: 0.2828 - val_loss: 2.8688 -
val_sparse_categorical_accuracy: 0.0990
Epoch 2/25
125/125 [==============================] - 9s 70ms/step - loss: 1.7225 -
sparse_categorical_accuracy: 0.3750 - val_loss: 4.2745 -
val_sparse_categorical_accuracy: 0.1050
Epoch 3/25
125/125 [==============================] - 9s 71ms/step - loss: 1.5716 -
```

```
sparse_categorical_accuracy: 0.4300 - val_loss: 2.4747 -
val_sparse_categorical_accuracy: 0.2460
Epoch 4/25
125/125 [==============================] - 9s 70ms/step - loss: 1.4767 -
sparse_categorical_accuracy: 0.4723 - val_loss: 2.5736 -
val_sparse_categorical_accuracy: 0.2040
Epoch 5/25
125/125 [==============================] - 9s 71ms/step - loss: 1.3580 -
sparse_categorical_accuracy: 0.5095 - val_loss: 2.5524 -
val_sparse_categorical_accuracy: 0.2940
Epoch 6/25
125/125 [==============================] - 9s 71ms/step - loss: 1.2302 -
sparse_categorical_accuracy: 0.5545 - val_loss: 1.7555 -
val_sparse_categorical_accuracy: 0.4330
Epoch 7/25
125/125 [==============================] - 9s 72ms/step - loss: 1.1133 -
sparse_categorical_accuracy: 0.5968 - val_loss: 2.3706 -
val_sparse_categorical_accuracy: 0.3390
Epoch 8/25
125/125 [==============================] - 9s 73ms/step - loss: 1.0124 -
sparse_categorical_accuracy: 0.6470 - val_loss: 1.8810 -
val_sparse_categorical_accuracy: 0.4050
Epoch 9/25
125/125 [==============================] - 9s 72ms/step - loss: 0.8895 -
sparse_categorical_accuracy: 0.6888 - val_loss: 2.2051 -
val_sparse_categorical_accuracy: 0.3710
Epoch 10/25
125/125 [==============================] - 9s 71ms/step - loss: 0.8074 -
sparse_categorical_accuracy: 0.7145 - val_loss: 1.8965 -
val_sparse_categorical_accuracy: 0.4610
Epoch 11/25
125/125 [==============================] - 9s 71ms/step - loss: 0.6483 -
sparse_categorical_accuracy: 0.7812 - val_loss: 1.7754 -
val_sparse_categorical_accuracy: 0.4660
Epoch 12/25
125/125 [==============================] - 9s 71ms/step - loss: 0.5829 -
sparse_categorical_accuracy: 0.7950 - val_loss: 2.4877 -
val_sparse_categorical_accuracy: 0.4430
Epoch 13/25
125/125 [==============================] - 9s 72ms/step - loss: 0.5056 -
sparse_categorical_accuracy: 0.8205 - val_loss: 2.0643 -
val_sparse_categorical_accuracy: 0.4570
Epoch 14/25
125/125 [==============================] - 9s 71ms/step - loss: 0.4214 -
sparse_categorical_accuracy: 0.8545 - val_loss: 1.9634 -
val_sparse_categorical_accuracy: 0.4640
Epoch 15/25
125/125 [==============================] - 9s 71ms/step - loss: 0.3868 -
```

```
sparse_categorical_accuracy: 0.8637 - val_loss: 2.3770 -
val_sparse_categorical_accuracy: 0.4690
Epoch 16/25
125/125 [==============================] - 9s 71ms/step - loss: 0.2893 -
sparse_categorical_accuracy: 0.9072 - val_loss: 2.9759 -
val_sparse_categorical_accuracy: 0.4490
Epoch 17/25
125/125 [==============================] - 9s 71ms/step - loss: 0.2780 -
sparse_categorical_accuracy: 0.9087 - val_loss: 2.1573 -
val_sparse_categorical_accuracy: 0.4710
Epoch 18/25
125/125 [==============================] - 9s 71ms/step - loss: 0.2481 -
sparse_categorical_accuracy: 0.9122 - val_loss: 2.6959 -
val_sparse_categorical_accuracy: 0.4170
Epoch 19/25
125/125 [==============================] - 9s 70ms/step - loss: 0.2462 -
sparse_categorical_accuracy: 0.9185 - val_loss: 2.4046 -
val_sparse_categorical_accuracy: 0.4810
Epoch 20/25
125/125 [==============================] - 9s 71ms/step - loss: 0.2137 -
sparse_categorical_accuracy: 0.9245 - val_loss: 2.3488 -
val_sparse_categorical_accuracy: 0.4810
Epoch 21/25
125/125 [==============================] - 9s 70ms/step - loss: 0.1661 -
sparse_categorical_accuracy: 0.9383 - val_loss: 2.6833 -
val_sparse_categorical_accuracy: 0.4970
Epoch 22/25
125/125 [==============================] - 9s 70ms/step - loss: 0.1896 -
sparse_categorical_accuracy: 0.9335 - val_loss: 2.3006 -
val_sparse_categorical_accuracy: 0.5150
Epoch 23/25
125/125 [==============================] - 9s 70ms/step - loss: 0.1377 -
sparse_categorical_accuracy: 0.9560 - val_loss: 2.4011 -
val_sparse_categorical_accuracy: 0.5130
Epoch 24/25
125/125 [==============================] - 9s 70ms/step - loss: 0.0995 -
sparse_categorical_accuracy: 0.9680 - val_loss: 2.7270 -
val_sparse_categorical_accuracy: 0.4760
Epoch 25/25
125/125 [==============================] - 9s 71ms/step - loss: 0.1490 -
sparse_categorical_accuracy: 0.9480 - val_loss: 3.7977 -
val_sparse_categorical_accuracy: 0.4140
```

[13]:

```
# Fitting the 50% Model

model_50.fit(x_train50,y_train50,epochs=25, batch_size=32, validation_split=0.2)
```

```
#history50 = model_50.fit(x_train50,y_train50,epochs=25, batch_size=32,
→validation_split=0.2)
```

```
Epoch 1/25
625/625 [==============================] - 44s 71ms/step - loss: 1.2155 -
sparse_categorical_accuracy: 0.5900 - val_loss: 1.2808 -
val_sparse_categorical_accuracy: 0.5534
Epoch 2/25
625/625 [==============================] - 45s 71ms/step - loss: 0.9159 -
sparse_categorical_accuracy: 0.6811 - val_loss: 0.9580 -
val_sparse_categorical_accuracy: 0.6660
Epoch 3/25
625/625 [==============================] - 44s 71ms/step - loss: 0.7727 -
sparse_categorical_accuracy: 0.7311 - val_loss: 1.2955 -
val_sparse_categorical_accuracy: 0.5846
Epoch 4/25
625/625 [==============================] - 44s 71ms/step - loss: 0.6411 -
sparse_categorical_accuracy: 0.7758 - val_loss: 1.2529 -
val_sparse_categorical_accuracy: 0.6184
Epoch 5/25
625/625 [==============================] - 44s 71ms/step - loss: 0.5366 -
sparse_categorical_accuracy: 0.8106 - val_loss: 0.9847 -
val_sparse_categorical_accuracy: 0.6828
Epoch 6/25
625/625 [==============================] - 44s 71ms/step - loss: 0.4508 -
sparse_categorical_accuracy: 0.8418 - val_loss: 1.6166 -
val_sparse_categorical_accuracy: 0.6036
Epoch 7/25
625/625 [==============================] - 44s 71ms/step - loss: 0.3576 -
sparse_categorical_accuracy: 0.8761 - val_loss: 1.4494 -
val_sparse_categorical_accuracy: 0.6480
Epoch 8/25
625/625 [==============================] - 44s 71ms/step - loss: 0.2906 -
sparse_categorical_accuracy: 0.8967 - val_loss: 1.2609 -
val_sparse_categorical_accuracy: 0.6894
Epoch 9/25
625/625 [==============================] - 44s 71ms/step - loss: 0.2490 -
sparse_categorical_accuracy: 0.9145 - val_loss: 1.5164 -
val_sparse_categorical_accuracy: 0.6356
Epoch 10/25
625/625 [==============================] - 44s 71ms/step - loss: 0.2144 -
sparse_categorical_accuracy: 0.9244 - val_loss: 1.0207 -
val_sparse_categorical_accuracy: 0.7328
Epoch 11/25
625/625 [==============================] - 43s 68ms/step - loss: 0.1743 -
sparse_categorical_accuracy: 0.9399 - val_loss: 1.6696 -
val_sparse_categorical_accuracy: 0.6480
```

```
Epoch 12/25
625/625 [==============================] - 41s 66ms/step - loss: 0.1651 -
sparse_categorical_accuracy: 0.9434 - val_loss: 1.3776 -
val_sparse_categorical_accuracy: 0.6784
Epoch 13/25
625/625 [==============================] - 41s 66ms/step - loss: 0.1524 -
sparse_categorical_accuracy: 0.9476 - val_loss: 1.2679 -
val_sparse_categorical_accuracy: 0.7086
Epoch 14/25
625/625 [==============================] - 41s 65ms/step - loss: 0.1390 -
sparse_categorical_accuracy: 0.9519 - val_loss: 1.0181 -
val_sparse_categorical_accuracy: 0.7444
Epoch 15/25
625/625 [==============================] - 40s 65ms/step - loss: 0.1237 -
sparse_categorical_accuracy: 0.9572 - val_loss: 1.4486 -
val_sparse_categorical_accuracy: 0.7046
Epoch 16/25
625/625 [==============================] - 41s 65ms/step - loss: 0.1141 -
sparse_categorical_accuracy: 0.9612 - val_loss: 1.6245 -
val_sparse_categorical_accuracy: 0.6578
Epoch 17/25
625/625 [==============================] - 41s 66ms/step - loss: 0.1104 -
sparse_categorical_accuracy: 0.9623 - val_loss: 1.5455 -
val_sparse_categorical_accuracy: 0.6826
Epoch 18/25
625/625 [==============================] - 46s 73ms/step - loss: 0.1010 -
sparse_categorical_accuracy: 0.9642 - val_loss: 1.4584 -
val_sparse_categorical_accuracy: 0.7050
Epoch 19/25
625/625 [==============================] - 48s 78ms/step - loss: 0.0948 -
sparse_categorical_accuracy: 0.9673 - val_loss: 1.3788 -
val_sparse_categorical_accuracy: 0.7264
Epoch 20/25
625/625 [==============================] - 49s 78ms/step - loss: 0.0886 -
sparse_categorical_accuracy: 0.9711 - val_loss: 1.5430 -
val_sparse_categorical_accuracy: 0.7002
Epoch 21/25
625/625 [==============================] - 49s 78ms/step - loss: 0.0925 -
sparse_categorical_accuracy: 0.9672 - val_loss: 1.7258 -
val_sparse_categorical_accuracy: 0.6658
Epoch 22/25
625/625 [==============================] - 48s 77ms/step - loss: 0.0803 -
sparse_categorical_accuracy: 0.9729 - val_loss: 1.2996 -
val_sparse_categorical_accuracy: 0.7270
Epoch 23/25
625/625 [==============================] - 48s 77ms/step - loss: 0.0820 -
sparse_categorical_accuracy: 0.9722 - val_loss: 1.3716 -
val_sparse_categorical_accuracy: 0.7294
```

```
Epoch 24/25
625/625 [==============================] - 49s 78ms/step - loss: 0.0835 -
sparse_categorical_accuracy: 0.9714 - val_loss: 1.6355 -
val_sparse_categorical_accuracy: 0.6962
Epoch 25/25
625/625 [==============================] - 48s 77ms/step - loss: 0.0662 -
sparse_categorical_accuracy: 0.9772 - val_loss: 1.6322 -
val_sparse_categorical_accuracy: 0.7040
```

[14]:

```
# Fitting the 80% Model

model_80.fit(x_train80,y_train80,epochs=25, batch_size=32, validation_split=0.2)
#history80 = model_80.fit(x_train80,y_train80,epochs=25, batch_size=32,␣
↪validation_split=0.2)
```

```
Epoch 1/25
1000/1000 [==============================] - 75s 75ms/step - loss: 0.5142 -
sparse_categorical_accuracy: 0.8465 - val_loss: 0.5194 -
val_sparse_categorical_accuracy: 0.8328
Epoch 2/25
1000/1000 [==============================] - 66s 66ms/step - loss: 0.2670 -
sparse_categorical_accuracy: 0.9132 - val_loss: 0.8483 -
val_sparse_categorical_accuracy: 0.7550
Epoch 3/25
1000/1000 [==============================] - 66s 66ms/step - loss: 0.1871 -
sparse_categorical_accuracy: 0.9354 - val_loss: 1.0471 -
val_sparse_categorical_accuracy: 0.7290
Epoch 4/25
1000/1000 [==============================] - 65s 65ms/step - loss: 0.1455 -
sparse_categorical_accuracy: 0.9506 - val_loss: 0.7504 -
val_sparse_categorical_accuracy: 0.7970
Epoch 5/25
1000/1000 [==============================] - 68s 68ms/step - loss: 0.1210 -
sparse_categorical_accuracy: 0.9590 - val_loss: 1.0309 -
val_sparse_categorical_accuracy: 0.7558
Epoch 6/25
1000/1000 [==============================] - 65s 65ms/step - loss: 0.1163 -
sparse_categorical_accuracy: 0.9593 - val_loss: 0.9084 -
val_sparse_categorical_accuracy: 0.7846
Epoch 7/25
1000/1000 [==============================] - 66s 66ms/step - loss: 0.0912 -
sparse_categorical_accuracy: 0.9685 - val_loss: 0.8923 -
val_sparse_categorical_accuracy: 0.7968
Epoch 8/25
1000/1000 [==============================] - 67s 67ms/step - loss: 0.0928 -
sparse_categorical_accuracy: 0.9678 - val_loss: 1.2350 -
val_sparse_categorical_accuracy: 0.7335
Epoch 9/25
```

```
1000/1000 [==============================] - 81s 81ms/step - loss: 0.0869 -
sparse_categorical_accuracy: 0.9699 - val_loss: 0.9269 -
val_sparse_categorical_accuracy: 0.7884
Epoch 10/25
1000/1000 [==============================] - 71s 71ms/step - loss: 0.0770 -
sparse_categorical_accuracy: 0.9734 - val_loss: 0.8906 -
val_sparse_categorical_accuracy: 0.8089
Epoch 11/25
1000/1000 [==============================] - 66s 66ms/step - loss: 0.0808 -
sparse_categorical_accuracy: 0.9729 - val_loss: 0.8408 -
val_sparse_categorical_accuracy: 0.8062
Epoch 12/25
1000/1000 [==============================] - 65s 65ms/step - loss: 0.0726 -
sparse_categorical_accuracy: 0.9751 - val_loss: 0.9230 -
val_sparse_categorical_accuracy: 0.7937
Epoch 13/25
1000/1000 [==============================] - 65s 65ms/step - loss: 0.0705 -
sparse_categorical_accuracy: 0.9759 - val_loss: 0.9507 -
val_sparse_categorical_accuracy: 0.7881
Epoch 14/25
1000/1000 [==============================] - 64s 64ms/step - loss: 0.0664 -
sparse_categorical_accuracy: 0.9771 - val_loss: 1.1564 -
val_sparse_categorical_accuracy: 0.7659
Epoch 15/25
1000/1000 [==============================] - 65s 65ms/step - loss: 0.0567 -
sparse_categorical_accuracy: 0.9803 - val_loss: 1.0516 -
val_sparse_categorical_accuracy: 0.7989
Epoch 16/25
1000/1000 [==============================] - 65s 65ms/step - loss: 0.0677 -
sparse_categorical_accuracy: 0.9764 - val_loss: 1.2900 -
val_sparse_categorical_accuracy: 0.7604
Epoch 17/25
1000/1000 [==============================] - 65s 65ms/step - loss: 0.0501 -
sparse_categorical_accuracy: 0.9827 - val_loss: 1.0839 -
val_sparse_categorical_accuracy: 0.7843
Epoch 18/25
1000/1000 [==============================] - 64s 64ms/step - loss: 0.0643 -
sparse_categorical_accuracy: 0.9784 - val_loss: 1.0430 -
val_sparse_categorical_accuracy: 0.7824
Epoch 19/25
1000/1000 [==============================] - 67s 67ms/step - loss: 0.0547 -
sparse_categorical_accuracy: 0.9817 - val_loss: 1.1560 -
val_sparse_categorical_accuracy: 0.7794
Epoch 20/25
1000/1000 [==============================] - 65s 65ms/step - loss: 0.0513 -
sparse_categorical_accuracy: 0.9825 - val_loss: 1.0055 -
val_sparse_categorical_accuracy: 0.7976
Epoch 21/25
```

```
1000/1000 [==============================] - 65s 65ms/step - loss: 0.0479 -
sparse_categorical_accuracy: 0.9839 - val_loss: 1.0299 -
val_sparse_categorical_accuracy: 0.7983
Epoch 22/25
1000/1000 [==============================] - 64s 64ms/step - loss: 0.0536 -
sparse_categorical_accuracy: 0.9824 - val_loss: 1.0672 -
val_sparse_categorical_accuracy: 0.7829
Epoch 23/25
1000/1000 [==============================] - 64s 64ms/step - loss: 0.0427 -
sparse_categorical_accuracy: 0.9862 - val_loss: 1.1263 -
val_sparse_categorical_accuracy: 0.7832
Epoch 24/25
1000/1000 [==============================] - 64s 64ms/step - loss: 0.0510 -
sparse_categorical_accuracy: 0.9829 - val_loss: 1.1432 -
val_sparse_categorical_accuracy: 0.7861
Epoch 25/25
1000/1000 [==============================] - 65s 65ms/step - loss: 0.0431 -
sparse_categorical_accuracy: 0.9849 - val_loss: 1.1585 -
val_sparse_categorical_accuracy: 0.7844
```

1.6 Plotting the Models

[72]:
```python
# Test Error

temp={}
temp1={}
temp2={}

score = model_10.evaluate(x_test, y_test, verbose=0)
temp["DenseNet121 10% Test error"] = 1-score[1]
performance = pd.DataFrame([temp]).T

score = model_50.evaluate(x_test, y_test, verbose=0)
temp1["DenseNet121 50% Test error"]=1-score[1]
performance1 = pd.DataFrame([temp1]).T
performance = performance.append(performance1)

score = model_80.evaluate(x_test, y_test, verbose=0)
temp2["DenseNet121 80% Test error"]=1-score[1]
performance2 = pd.DataFrame([temp2]).T
performance = performance.append(performance2)

performance
```

[72]:

	0
DenseNet121 10% Test error	0.2593
DenseNet121 50% Test error	0.2593

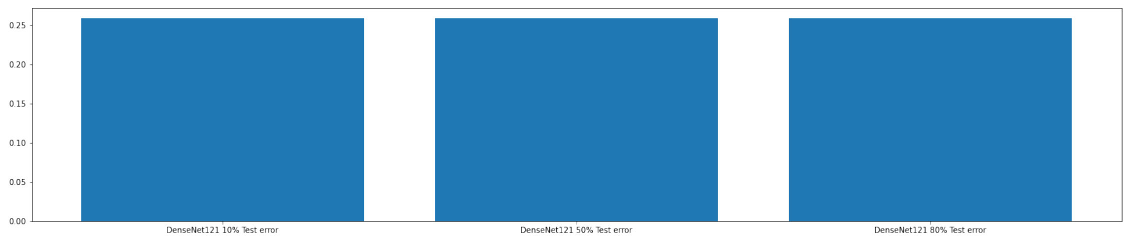
```
DenseNet121 80% Test error  0.2593
```

[77]:
```python
# Plotting Bar Plot

names = ('DenseNet121 10% Test error', 'DenseNet121 50% Test error',
         'DenseNet121 80% Test error')

values = (0.2593,0.2593,0.2593)

fig = plt.figure(figsize = (25,5))
plt.bar(names, values)
plt.show()
```



1.6.1 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()

1.6.2 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()

1.6.3 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()

2 Homework 2 My Model

2.1 Setting up the Model

[27]:
```python
# Designing the Model


custom_model=tf.keras.models.Sequential()
```

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

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

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

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

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

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

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


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

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

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

custom_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 32, 32, 32)        896

 max_pooling2d_4 (MaxPooling  (None, 16, 16, 32)       0
 2D)
```

```
 conv2d_5 (Conv2D)           (None, 16, 16, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 flatten_2 (Flatten)         (None, 4096)              0

 dropout_2 (Dropout)         (None, 4096)              0

 dense_4 (Dense)             (None, 128)               524416

 dense_5 (Dense)             (None, 10)                1290

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

[29]:
```
# Setting up the Different Versions

custom_model_10 = custom_model
custom_model_50 = custom_model
custom_model_80 = custom_model
```

[30]:
```
# Compiling Models

custom_model_10.compile(loss="sparse_categorical_crossentropy",
                      optimizer="Adam", metrics=["sparse_categorical_accuracy"])

custom_model_50.compile(loss="sparse_categorical_crossentropy",
                      optimizer="Adam", metrics=["sparse_categorical_accuracy"])

custom_model_80.compile(loss="sparse_categorical_crossentropy",
                      optimizer="Adam", metrics=["sparse_categorical_accuracy"])
```

2.2 Training My Models

[43]:
```
# Train 10% Model

custom_model_10.fit(x_train10,y_train10,epochs=25,batch_size=32,␣
 ↪validation_split=0.2)
#history10_2 = custom_model_10.fit(x_train10,y_train10,epochs=25,batch_size=32,␣
 ↪validation_split=0.2)
```

```
Epoch 1/25
125/125 [==============================] - 1s 6ms/step - loss: 1.2872 -
```

```
sparse_categorical_accuracy: 0.6367 - val_loss: 0.9469 -
val_sparse_categorical_accuracy: 0.6790
Epoch 2/25
125/125 [==============================] - 0s 4ms/step - loss: 0.9378 -
sparse_categorical_accuracy: 0.6905 - val_loss: 0.9234 -
val_sparse_categorical_accuracy: 0.6890
Epoch 3/25
125/125 [==============================] - 0s 4ms/step - loss: 0.7558 -
sparse_categorical_accuracy: 0.7442 - val_loss: 0.8971 -
val_sparse_categorical_accuracy: 0.7130
Epoch 4/25
125/125 [==============================] - 1s 4ms/step - loss: 0.6541 -
sparse_categorical_accuracy: 0.7740 - val_loss: 0.9213 -
val_sparse_categorical_accuracy: 0.6970
Epoch 5/25
125/125 [==============================] - 0s 4ms/step - loss: 0.5543 -
sparse_categorical_accuracy: 0.8012 - val_loss: 0.9280 -
val_sparse_categorical_accuracy: 0.7070
Epoch 6/25
125/125 [==============================] - 0s 4ms/step - loss: 0.4811 -
sparse_categorical_accuracy: 0.8267 - val_loss: 0.9431 -
val_sparse_categorical_accuracy: 0.7010
Epoch 7/25
125/125 [==============================] - 0s 4ms/step - loss: 0.3989 -
sparse_categorical_accuracy: 0.8575 - val_loss: 0.9496 -
val_sparse_categorical_accuracy: 0.7000
Epoch 8/25
125/125 [==============================] - 0s 4ms/step - loss: 0.3476 -
sparse_categorical_accuracy: 0.8723 - val_loss: 1.0057 -
val_sparse_categorical_accuracy: 0.7000
Epoch 9/25
125/125 [==============================] - 1s 4ms/step - loss: 0.3260 -
sparse_categorical_accuracy: 0.8827 - val_loss: 0.9781 -
val_sparse_categorical_accuracy: 0.7170
Epoch 10/25
125/125 [==============================] - 1s 4ms/step - loss: 0.2742 -
sparse_categorical_accuracy: 0.9107 - val_loss: 1.0438 -
val_sparse_categorical_accuracy: 0.6970
Epoch 11/25
125/125 [==============================] - 1s 4ms/step - loss: 0.2233 -
sparse_categorical_accuracy: 0.9185 - val_loss: 1.0469 -
val_sparse_categorical_accuracy: 0.7060
Epoch 12/25
125/125 [==============================] - 1s 4ms/step - loss: 0.2328 -
sparse_categorical_accuracy: 0.9193 - val_loss: 1.0901 -
val_sparse_categorical_accuracy: 0.7010
Epoch 13/25
125/125 [==============================] - 1s 4ms/step - loss: 0.1920 -
```

```
sparse_categorical_accuracy: 0.9330 - val_loss: 1.1185 -
val_sparse_categorical_accuracy: 0.6970
Epoch 14/25
125/125 [==============================] - 1s 4ms/step - loss: 0.1814 -
sparse_categorical_accuracy: 0.9408 - val_loss: 1.1440 -
val_sparse_categorical_accuracy: 0.7160
Epoch 15/25
125/125 [==============================] - 1s 4ms/step - loss: 0.1754 -
sparse_categorical_accuracy: 0.9340 - val_loss: 1.1586 -
val_sparse_categorical_accuracy: 0.7010
Epoch 16/25
125/125 [==============================] - 1s 4ms/step - loss: 0.1462 -
sparse_categorical_accuracy: 0.9498 - val_loss: 1.1533 -
val_sparse_categorical_accuracy: 0.7130
Epoch 17/25
125/125 [==============================] - 1s 4ms/step - loss: 0.1437 -
sparse_categorical_accuracy: 0.9525 - val_loss: 1.1719 -
val_sparse_categorical_accuracy: 0.7100
Epoch 18/25
125/125 [==============================] - 1s 4ms/step - loss: 0.1221 -
sparse_categorical_accuracy: 0.9610 - val_loss: 1.2084 -
val_sparse_categorical_accuracy: 0.7020
Epoch 19/25
125/125 [==============================] - 1s 4ms/step - loss: 0.1023 -
sparse_categorical_accuracy: 0.9635 - val_loss: 1.1852 -
val_sparse_categorical_accuracy: 0.7040
Epoch 20/25
125/125 [==============================] - 1s 4ms/step - loss: 0.1044 -
sparse_categorical_accuracy: 0.9655 - val_loss: 1.2216 -
val_sparse_categorical_accuracy: 0.7000
Epoch 21/25
125/125 [==============================] - 1s 4ms/step - loss: 0.0991 -
sparse_categorical_accuracy: 0.9660 - val_loss: 1.2077 -
val_sparse_categorical_accuracy: 0.7090
Epoch 22/25
125/125 [==============================] - 1s 4ms/step - loss: 0.0892 -
sparse_categorical_accuracy: 0.9700 - val_loss: 1.3256 -
val_sparse_categorical_accuracy: 0.7020
Epoch 23/25
125/125 [==============================] - 1s 4ms/step - loss: 0.1025 -
sparse_categorical_accuracy: 0.9647 - val_loss: 1.2640 -
val_sparse_categorical_accuracy: 0.7000
Epoch 24/25
125/125 [==============================] - 1s 4ms/step - loss: 0.0845 -
sparse_categorical_accuracy: 0.9730 - val_loss: 1.2801 -
val_sparse_categorical_accuracy: 0.6930
Epoch 25/25
125/125 [==============================] - 1s 4ms/step - loss: 0.0953 -
```

```
sparse_categorical_accuracy: 0.9685 - val_loss: 1.3376 -
val_sparse_categorical_accuracy: 0.6970
```

[43]: <keras.callbacks.History at 0x218474ee220>

[44]:
```python
# Train 50% Model

custom_model_50.fit(x_train50,y_train50,epochs=25,batch_size=32,␣
 ↪validation_split=0.2)
#history50_2 = custom_model_50.fit(x_train50,y_train50,epochs=25,batch_size=32,␣
 ↪validation_split=0.2)
```

```
Epoch 1/25
625/625 [==============================] - 3s 4ms/step - loss: 0.5159 -
sparse_categorical_accuracy: 0.8339 - val_loss: 0.2590 -
val_sparse_categorical_accuracy: 0.9354
Epoch 2/25
625/625 [==============================] - 2s 4ms/step - loss: 0.3501 -
sparse_categorical_accuracy: 0.8796 - val_loss: 0.2824 -
val_sparse_categorical_accuracy: 0.9204
Epoch 3/25
625/625 [==============================] - 2s 4ms/step - loss: 0.2727 -
sparse_categorical_accuracy: 0.9060 - val_loss: 0.2877 -
val_sparse_categorical_accuracy: 0.9198
Epoch 4/25
625/625 [==============================] - 2s 4ms/step - loss: 0.2629 -
sparse_categorical_accuracy: 0.9096 - val_loss: 0.3217 -
val_sparse_categorical_accuracy: 0.9064
Epoch 5/25
625/625 [==============================] - 2s 4ms/step - loss: 0.2354 -
sparse_categorical_accuracy: 0.9194 - val_loss: 0.3758 -
val_sparse_categorical_accuracy: 0.8918
Epoch 6/25
625/625 [==============================] - 2s 4ms/step - loss: 0.2147 -
sparse_categorical_accuracy: 0.9256 - val_loss: 0.3469 -
val_sparse_categorical_accuracy: 0.9034
Epoch 7/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1966 -
sparse_categorical_accuracy: 0.9323 - val_loss: 0.3592 -
val_sparse_categorical_accuracy: 0.8998
Epoch 8/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1950 -
sparse_categorical_accuracy: 0.9311 - val_loss: 0.3576 -
val_sparse_categorical_accuracy: 0.9018
Epoch 9/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1853 -
sparse_categorical_accuracy: 0.9366 - val_loss: 0.3872 -
val_sparse_categorical_accuracy: 0.8884
```

```
Epoch 10/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1787 -
sparse_categorical_accuracy: 0.9395 - val_loss: 0.4205 -
val_sparse_categorical_accuracy: 0.8834
Epoch 11/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1662 -
sparse_categorical_accuracy: 0.9415 - val_loss: 0.4134 -
val_sparse_categorical_accuracy: 0.8834
Epoch 12/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1637 -
sparse_categorical_accuracy: 0.9438 - val_loss: 0.4628 -
val_sparse_categorical_accuracy: 0.8670
Epoch 13/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1680 -
sparse_categorical_accuracy: 0.9429 - val_loss: 0.4705 -
val_sparse_categorical_accuracy: 0.8676
Epoch 14/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1443 -
sparse_categorical_accuracy: 0.9506 - val_loss: 0.4628 -
val_sparse_categorical_accuracy: 0.8692
Epoch 15/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1492 -
sparse_categorical_accuracy: 0.9491 - val_loss: 0.4922 -
val_sparse_categorical_accuracy: 0.8694
Epoch 16/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1535 -
sparse_categorical_accuracy: 0.9488 - val_loss: 0.4892 -
val_sparse_categorical_accuracy: 0.8592
Epoch 17/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1480 -
sparse_categorical_accuracy: 0.9503 - val_loss: 0.5376 -
val_sparse_categorical_accuracy: 0.8528
Epoch 18/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1468 -
sparse_categorical_accuracy: 0.9505 - val_loss: 0.5298 -
val_sparse_categorical_accuracy: 0.8566
Epoch 19/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1461 -
sparse_categorical_accuracy: 0.9504 - val_loss: 0.5401 -
val_sparse_categorical_accuracy: 0.8518
Epoch 20/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1348 -
sparse_categorical_accuracy: 0.9566 - val_loss: 0.5582 -
val_sparse_categorical_accuracy: 0.8480
Epoch 21/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1304 -
sparse_categorical_accuracy: 0.9542 - val_loss: 0.5386 -
val_sparse_categorical_accuracy: 0.8540
```

```
Epoch 22/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1315 -
sparse_categorical_accuracy: 0.9560 - val_loss: 0.5511 -
val_sparse_categorical_accuracy: 0.8472
Epoch 23/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1204 -
sparse_categorical_accuracy: 0.9589 - val_loss: 0.6291 -
val_sparse_categorical_accuracy: 0.8356
Epoch 24/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1449 -
sparse_categorical_accuracy: 0.9506 - val_loss: 0.6171 -
val_sparse_categorical_accuracy: 0.8400
Epoch 25/25
625/625 [==============================] - 2s 4ms/step - loss: 0.1272 -
sparse_categorical_accuracy: 0.9565 - val_loss: 0.6487 -
val_sparse_categorical_accuracy: 0.8334
```

[44]: <keras.callbacks.History at 0x218450d5a30>

[45]:
```
# Train 80% Model

custom_model_80.fit(x_train80,y_train80,epochs=25,batch_size=32,␣
 ↪validation_split=0.2)
#history80_2 = custom_model_80.fit(x_train80,y_train80,epochs=25,batch_size=32,␣
 ↪validation_split=0.2)
```

```
Epoch 1/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.4244 -
sparse_categorical_accuracy: 0.8681 - val_loss: 0.3612 -
val_sparse_categorical_accuracy: 0.8870
Epoch 2/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.3340 -
sparse_categorical_accuracy: 0.8895 - val_loss: 0.3338 -
val_sparse_categorical_accuracy: 0.8940
Epoch 3/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.2856 -
sparse_categorical_accuracy: 0.9054 - val_loss: 0.3433 -
val_sparse_categorical_accuracy: 0.8953
Epoch 4/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.2596 -
sparse_categorical_accuracy: 0.9113 - val_loss: 0.3610 -
val_sparse_categorical_accuracy: 0.8848
Epoch 5/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.2546 -
sparse_categorical_accuracy: 0.9133 - val_loss: 0.3710 -
val_sparse_categorical_accuracy: 0.8880
Epoch 6/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.2404 -
```

```
sparse_categorical_accuracy: 0.9193 - val_loss: 0.3735 -
val_sparse_categorical_accuracy: 0.8824
Epoch 7/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.2267 -
sparse_categorical_accuracy: 0.9217 - val_loss: 0.3778 -
val_sparse_categorical_accuracy: 0.8824
Epoch 8/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.2214 -
sparse_categorical_accuracy: 0.9245 - val_loss: 0.4336 -
val_sparse_categorical_accuracy: 0.8671
Epoch 9/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.2185 -
sparse_categorical_accuracy: 0.9245 - val_loss: 0.4082 -
val_sparse_categorical_accuracy: 0.8791
Epoch 10/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.2054 -
sparse_categorical_accuracy: 0.9287 - val_loss: 0.4179 -
val_sparse_categorical_accuracy: 0.8769
Epoch 11/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.1999 -
sparse_categorical_accuracy: 0.9314 - val_loss: 0.4377 -
val_sparse_categorical_accuracy: 0.8705
Epoch 12/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.1924 -
sparse_categorical_accuracy: 0.9340 - val_loss: 0.4284 -
val_sparse_categorical_accuracy: 0.8716
Epoch 13/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.1959 -
sparse_categorical_accuracy: 0.9334 - val_loss: 0.4748 -
val_sparse_categorical_accuracy: 0.8604
Epoch 14/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.1901 -
sparse_categorical_accuracy: 0.9340 - val_loss: 0.4404 -
val_sparse_categorical_accuracy: 0.8691
Epoch 15/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.1937 -
sparse_categorical_accuracy: 0.9340 - val_loss: 0.4960 -
val_sparse_categorical_accuracy: 0.8543
Epoch 16/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.1882 -
sparse_categorical_accuracy: 0.9357 - val_loss: 0.5059 -
val_sparse_categorical_accuracy: 0.8489
Epoch 17/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.1805 -
sparse_categorical_accuracy: 0.9381 - val_loss: 0.5387 -
val_sparse_categorical_accuracy: 0.8505
Epoch 18/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.1726 -
```

```
sparse_categorical_accuracy: 0.9401 - val_loss: 0.4898 -
val_sparse_categorical_accuracy: 0.8520
Epoch 19/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.1765 -
sparse_categorical_accuracy: 0.9399 - val_loss: 0.5162 -
val_sparse_categorical_accuracy: 0.8471
Epoch 20/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.1787 -
sparse_categorical_accuracy: 0.9390 - val_loss: 0.5187 -
val_sparse_categorical_accuracy: 0.8470
Epoch 21/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.1738 -
sparse_categorical_accuracy: 0.9412 - val_loss: 0.5559 -
val_sparse_categorical_accuracy: 0.8439
Epoch 22/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.1741 -
sparse_categorical_accuracy: 0.9408 - val_loss: 0.5191 -
val_sparse_categorical_accuracy: 0.8515
Epoch 23/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.1677 -
sparse_categorical_accuracy: 0.9434 - val_loss: 0.5385 -
val_sparse_categorical_accuracy: 0.8434
Epoch 24/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.1729 -
sparse_categorical_accuracy: 0.9417 - val_loss: 0.5775 -
val_sparse_categorical_accuracy: 0.8369
Epoch 25/25
1000/1000 [==============================] - 4s 4ms/step - loss: 0.1594 -
sparse_categorical_accuracy: 0.9455 - val_loss: 0.5809 -
val_sparse_categorical_accuracy: 0.8349
```

[45]: <keras.callbacks.History at 0x218450e7f40>

2.3 Plotting the Loss of My Model

[58]:
```
## Test Error



temp3={}
temp4={}
temp5={}

score = custom_model_10.evaluate(x_test, y_test, verbose=0)
temp3["Custom 10% Test error"]=1-score[1]
performance3 = pd.DataFrame([temp3]).T
performance = performance.append(performance3)
```

```
score2 = custom_model_50.evaluate(x_test, y_test, verbose=0)
temp4["Custom 50%  Test error"]=1-score[1]
performance4 = pd.DataFrame([temp4]).T
performance = performance.append(performance4)

score3 = custom_model_80.evaluate(x_test, y_test, verbose=0)
temp5["Custom 80%  Test error"]=1-score[1]
performance5 = pd.DataFrame([temp5]).T
performance = performance.append(performance5)

performance
```

[58]:

	0
DenseNet121 10% Test error	0.2593
DenseNet121 50% Test error	0.2593
DenseNet121 80% Test error	0.2593
Custom 10% Test error	0.2862
Custom 50% Test error	0.2862
Custom 80% Test error	0.2862

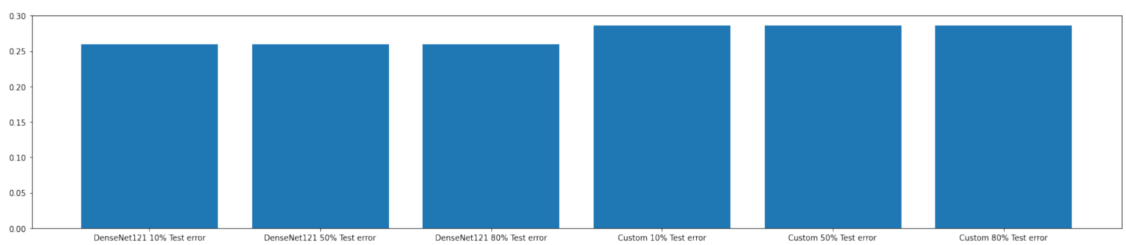
[76]:

```
# Plotting Bar Plot

names = ('DenseNet121 10% Test error', 'DenseNet121 50% Test error',
         'DenseNet121 80% Test error','Custom 10% Test error',
         'Custom 50% Test error','Custom 80% Test error')

values = (0.2593,0.2593,0.2593,0.2862,0.2862,0.2862)

fig = plt.figure(figsize = (25,5))
plt.bar(names, values)
plt.show()
```



3 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()

4 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()

5 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()