# Project02_Python

November 5, 2023

[1]:
```
# Road Map

#   Step 1  - EDA
#   Step 2  - Create our Custom Dataset with Labels
#   Step 3  - Split DataFrame into Test and Train
#   Step 4  - Create Project 2 CNN Model
#   Step 5  - Running a Test CNN to ensure model functions over 10 epochs
#   Step 6  - Use Optuna Optimizer on our CNN model and apply parameters over␣
 ↪100 epochs
#   Step 7  - Validate the Final model 10 times using 150 epochs
#   Step 8  - Run the final model over 500 epochs
#   Step 9  - Evaluate with "suitable metrics" (Confusion Matrix, AUCROC)
#   Step 10 - Save the Model
# X Step 11 - Write the technical report
# X Step 12 - Compile all documents
```

[2]:
```
# Put all packages here
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import torch
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score, average_precision_score
import torch.nn as nn
import torch.nn.functional as F
from flask import Flask, render_template, request, jsonify
import cv2
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader, random_split
from PIL import Image
from sklearn.model_selection import train_test_split
import os
import optuna
from sklearn.model_selection import StratifiedKFold
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import classification_report, accuracy_score
import statistics
import scipy.stats as st
import datetime
from IPython.display import clear_output
```

```
C:\Users\Garrett\anaconda3\envs\SD7502\lib\site-packages\tqdm\auto.py:21:
TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

[3]:
```
#Declaring all Project 2 Variables

# Dataset Variables
sizeClasses = 2
sizeBatch = 16
sizeChannel = 16

# Image Size
imageHeight = 7
imageWidth = 7
imageTransHeight = 9
imageTransWidth = 9

learningRate = 0.01
testSize = 0.2
randomState = 42

# CNN Variables
sizeKernel = 3
sizeStride = 1
sizePadding = 1
sizePoolKernel = 2
sizePoolStride = 2
testEpochs = 250
tuneEpochs = 250
validateEpochs = 250
finalEpochs = 250

# Import Data Sets
arrayLions = []
arrayCheetahs = []
folderLions = ".\\images\\Lions"
folderCheetahs = ".\\images\\Cheetahs"
folderRoot = ".\\images"
```

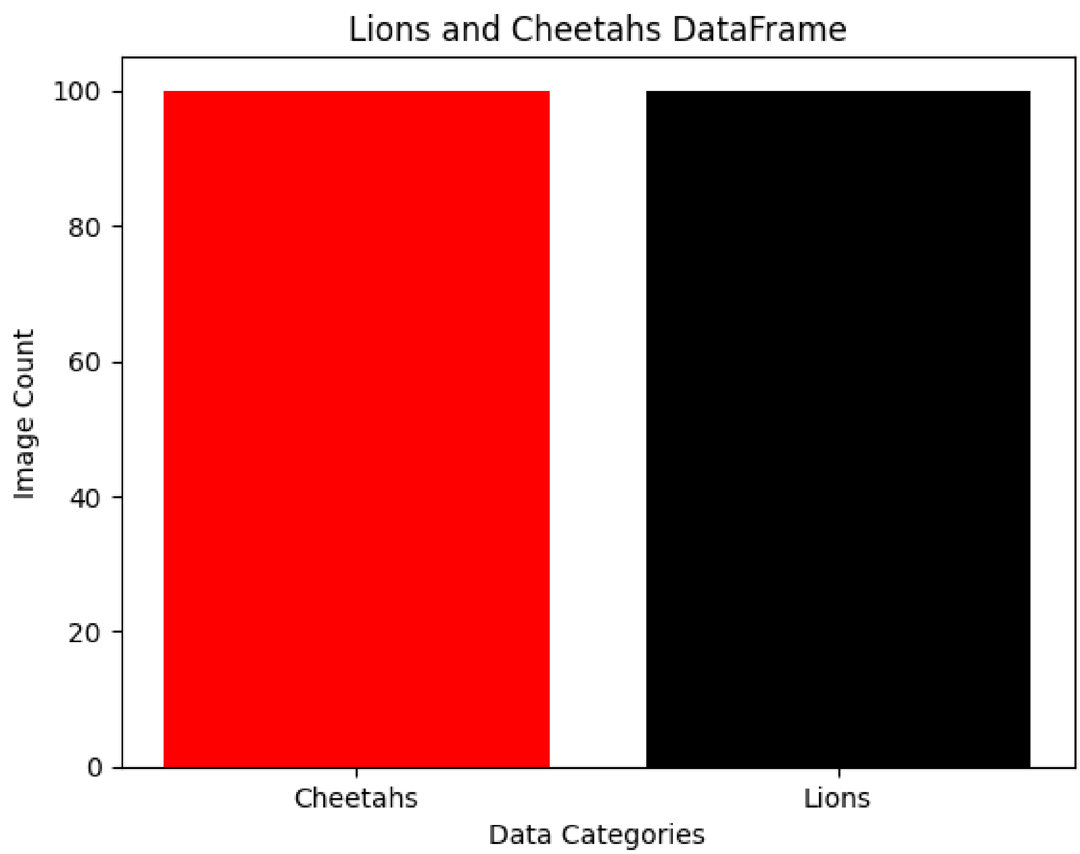
```
# K-Fold Validation
nSplits = 10
```

[4]:
```
# Creating the DataFrame

# Importing Lions
for r,d,f in os.walk(folderLions):
    for file in f:
        arrayLions.append((os.path.join(folderLions, file), "0"))

# Importing Cheetahs
for r,d,f in os.walk(folderCheetahs):
    for file in f:
        arrayCheetahs.append((os.path.join(folderCheetahs, file), "1"))

# # Combining Numpy Arrays
arrayCombined = np.concatenate((arrayLions, arrayCheetahs), axis = 0)

# # Creating the DataFrame
df = pd.DataFrame(arrayCombined, columns = ['name', 'label'])
```

[5]:
```
# Apply Labels to the DataFrame

# Setting the Transforms
transform = transforms.Compose([
                                transforms.Resize([imageTransHeight,␣
 ↪imageTransWidth]),
                                transforms.ToTensor(),
                                ])

# Label Mapping
mappingLabel = {'0': 0, '1': 1}

# Creating an LC Dataset of transformed images
class LCDataset(Dataset):
    def __init__(self, df, transform = None):
        self.df = df
        self.fileName = df['name'].values
        self.labels = df['label'].values
        self.transform = transform

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, index):
        pathImage = self.fileName[index]
        image = Image.open(pathImage).convert('RGB')
```

```
        if self.transform:
            image = self.transform(image)
        label = torch.tensor(mappingLabel[self.labels[index]],
                             dtype = torch.long)
        return image, label

# Loading the New Dataset
newDataset = LCDataset(df, transform = transform)
```

[6]:
```
# Project Two EDA

# Breakdown of number of labels
count = pd.DataFrame(df[["label"]].value_counts()).reset_index()["count"]

# Making Categories
categories = ['Cheetahs', 'Lions']

# Plotting the Data
plt.bar(categories, count, color=['red', 'black'])
plt.xlabel('Data Categories')
plt.ylabel('Image Count')
plt.title('Lions and Cheetahs DataFrame')
plt.show()
```




[7]:
```
# Creating the Test and Train Sets and DataLoaders

# Creating Train and Test Sets
setTrain, setTest = random_split(newDataset, [0.8, 0.2])

# Loading Datasets
trainLoader = torch.utils.data.DataLoader(setTrain, batch_size = sizeBatch,␣
 ↪shuffle = True)
testLoader = torch.utils.data.DataLoader(setTest, batch_size = sizeBatch,␣
 ↪shuffle = True)
```

[8]:
```
# Creating the CNN for Project 2

class proj2CNN(nn.Module):
    def __init__(self, sizeChannel, sizeKernel, sizeStride, sizePadding,␣
 ↪sizePoolKernel, sizePoolStride, imageHeight, imageWidth):
        super(proj2CNN, self).__init__()
        # 3 input channels, X output channels, 3*3 kernel, 1 pixel of padding
        self.conv1 = nn.Conv2d(3, sizeChannel, 3, padding=1)
```

```
        # X input channels, X * 2 output channels, 3*3 kernel, 1 pixel of␣
↪padding
        self.conv2 = nn.Conv2d(sizeChannel, sizeChannel * 2, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(sizeChannel * 2 * 2 * 2, 128)
        self.relu = nn.ReLU()
        # Model collapses down to one of two labels, 0 and 1
        self.fc2 = nn.Linear(128, 2)

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
#          This is used for finding the values for the view
#          print(x.size())
        x = x.view(-1, (sizeChannel* 2) * 2 * 2)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

[9]:
```
# Setting initial parameters for the test Model

# Creating an instance of proj2CNN
testProj2CNNetwork = proj2CNN(sizeChannel, sizeKernel, sizeStride, sizePadding,␣
↪sizePoolKernel, sizePoolStride, imageHeight, imageWidth)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(testProj2CNNetwork.parameters(), lr = learningRate)
```

[10]:
```
# Running the initial test of the CNN Model and initial validation

# Setting the traingLosses array for data storage
trainingLosses = []

# # Runs the models for the number of desired Epochs
for epoch in range(testEpochs):
    trainingLoss = 0.0
    for i, data in enumerate(trainLoader, 0):
        inputs, labels = data
        optimizer.zero_grad()
        outputs = testProj2CNNetwork(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        trainingLoss += loss.item()
    trainingLosses.append(trainingLoss / len(trainLoader))
    print(f"Epoch {epoch+1}, Loss: {trainingLoss / len(trainLoader)}")
```

```python
print("Testing Model Completed")

correct = 0
total = 0
with torch.no_grad():
    for data in testLoader:
        inputs, labels = data
        outputs = testProj2CNNetwork(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
print(f"Accuracy on the test dataset: {100 * correct / total}%")
```

```
Epoch 1, Loss: 0.7117283642292023
Epoch 2, Loss: 0.6948799014091491
Epoch 3, Loss: 0.6935420572757721
Epoch 4, Loss: 0.6932750821113587
Epoch 5, Loss: 0.6932962954044342
Epoch 6, Loss: 0.6941050708293914
Epoch 7, Loss: 0.6933500468730927
Epoch 8, Loss: 0.693219214677810
Epoch 9, Loss: 0.6935785710811615
Epoch 10, Loss: 0.6934297621250153
Epoch 11, Loss: 0.6930543184280396
Epoch 12, Loss: 0.6927850186824799
Epoch 13, Loss: 0.6930086195468903
Epoch 14, Loss: 0.6953892648220062
Epoch 15, Loss: 0.6920224010944367
Epoch 16, Loss: 0.6961346626281738
Epoch 17, Loss: 0.6944967210292816
Epoch 18, Loss: 0.6947400391101837
Epoch 19, Loss: 0.6934503018856049
Epoch 20, Loss: 0.6938700616359711
Epoch 21, Loss: 0.690564614534378
Epoch 22, Loss: 0.6918363451957703
Epoch 23, Loss: 0.6897874772548676
Epoch 24, Loss: 0.6720916926860809
Epoch 25, Loss: 0.6625658512115479
Epoch 26, Loss: 0.650997257232666
Epoch 27, Loss: 0.6247638821601867
Epoch 28, Loss: 0.6167818069458008
Epoch 29, Loss: 0.6203086495399475
Epoch 30, Loss: 0.5981501936912537
Epoch 31, Loss: 0.5714799910783768
Epoch 32, Loss: 0.542090654373169
Epoch 33, Loss: 0.5416574597358703
Epoch 34, Loss: 0.5469374179840087
Epoch 35, Loss: 0.529155108332634
```

```
Epoch 36, Loss: 0.5111190527677536
Epoch 37, Loss: 0.49873577058315277
Epoch 38, Loss: 0.5066328853368759
Epoch 39, Loss: 0.4308164894580841
Epoch 40, Loss: 0.44674046635627745
Epoch 41, Loss: 0.421274334192276
Epoch 42, Loss: 0.3754655793309212
Epoch 43, Loss: 0.3857862576842308
Epoch 44, Loss: 0.3439143419265747
Epoch 45, Loss: 0.34598884731531143
Epoch 46, Loss: 0.3904461745172739
Epoch 47, Loss: 0.35811128914356233
Epoch 48, Loss: 0.2940074816346169
Epoch 49, Loss: 0.27911261320114134
Epoch 50, Loss: 0.23856543004512787
Epoch 51, Loss: 0.2535006210207939
Epoch 52, Loss: 0.23854406252503396
Epoch 53, Loss: 0.19470086470246314
Epoch 54, Loss: 0.18815367221832274
Epoch 55, Loss: 0.1499046877026558
Epoch 56, Loss: 0.16300422251224517
Epoch 57, Loss: 0.15284814052283763
Epoch 58, Loss: 0.1807180318981409
Epoch 59, Loss: 0.10275369621813298
Epoch 60, Loss: 0.09316123314201832
Epoch 61, Loss: 0.10423238165676593
Epoch 62, Loss: 0.108107540756464
Epoch 63, Loss: 0.09086258476600051
Epoch 64, Loss: 0.09774872027337551
Epoch 65, Loss: 0.15921458825469018
Epoch 66, Loss: 0.19410034753382205
Epoch 67, Loss: 0.4096639212220907
Epoch 68, Loss: 0.4437733441591263
Epoch 69, Loss: 0.32705679833889006
Epoch 70, Loss: 0.23009923323988915
Epoch 71, Loss: 0.17245709523558617
Epoch 72, Loss: 0.12331742756068706
Epoch 73, Loss: 0.10976174585521221
Epoch 74, Loss: 0.07368751894682646
Epoch 75, Loss: 0.05799454636871815
Epoch 76, Loss: 0.03891198066994548
Epoch 77, Loss: 0.03984024506062269
Epoch 78, Loss: 0.024585549347102643
Epoch 79, Loss: 0.026840241532772778
Epoch 80, Loss: 0.019544290215708315
Epoch 81, Loss: 0.015414199652150273
Epoch 82, Loss: 0.012820601928979158
Epoch 83, Loss: 0.0104419173207134
```

```
Epoch 84, Loss: 0.008991757407784462
Epoch 85, Loss: 0.007912778039462864
Epoch 86, Loss: 0.006690854439511895
Epoch 87, Loss: 0.005705411487724632
Epoch 88, Loss: 0.0053463697666302325
Epoch 89, Loss: 0.0047316832933574915
Epoch 90, Loss: 0.004387633816804737
Epoch 91, Loss: 0.004272279737051576
Epoch 92, Loss: 0.0037134127574972807
Epoch 93, Loss: 0.003469748783390969
Epoch 94, Loss: 0.003237179049756378
Epoch 95, Loss: 0.0029785014339722693
Epoch 96, Loss: 0.002850297206896357
Epoch 97, Loss: 0.002616182784549892
Epoch 98, Loss: 0.0024587213294580577
Epoch 99, Loss: 0.0023691698763286693
Epoch 100, Loss: 0.0021818868932314216
Epoch 101, Loss: 0.002129575013532303
Epoch 102, Loss: 0.0020109888137085363
Epoch 103, Loss: 0.0020215493161231278
Epoch 104, Loss: 0.0018385739662335255
Epoch 105, Loss: 0.0017500714864581823
Epoch 106, Loss: 0.0016895658147404902
Epoch 107, Loss: 0.0016384746239054948
Epoch 108, Loss: 0.0015516142244450747
Epoch 109, Loss: 0.0015171844788710587
Epoch 110, Loss: 0.0014327524171676488
Epoch 111, Loss: 0.0013695807254407554
Epoch 112, Loss: 0.0013165645359549671
Epoch 113, Loss: 0.0012830392777686938
Epoch 114, Loss: 0.001272360229631886
Epoch 115, Loss: 0.0012219395895954222
Epoch 116, Loss: 0.0011472103680716828
Epoch 117, Loss: 0.001135778645402752
Epoch 118, Loss: 0.0011017473472747952
Epoch 119, Loss: 0.0010592110396828503
Epoch 120, Loss: 0.001055801275651902
Epoch 121, Loss: 0.0010085233574500308
Epoch 122, Loss: 0.0009599016921129078
Epoch 123, Loss: 0.0009299101686337963
Epoch 124, Loss: 0.0009125585260335356
Epoch 125, Loss: 0.0008818457077722997
Epoch 126, Loss: 0.0008519438619259745
Epoch 127, Loss: 0.0008270314865512773
Epoch 128, Loss: 0.0008057662023929879
Epoch 129, Loss: 0.0007837689059670083
Epoch 130, Loss: 0.0007676899724174291
Epoch 131, Loss: 0.0007437847583787516
```

```
Epoch 132, Loss: 0.0007265226944582537
Epoch 133, Loss: 0.0007221187377581372
Epoch 134, Loss: 0.0006963588151847943
Epoch 135, Loss: 0.000688238519069273
Epoch 136, Loss: 0.0006611897057155147
Epoch 137, Loss: 0.000657241404405795
Epoch 138, Loss: 0.0006558219931321219
Epoch 139, Loss: 0.0006222301279194653
Epoch 140, Loss: 0.0006019603941240347
Epoch 141, Loss: 0.0005844942148542032
Epoch 142, Loss: 0.0005868908672709949
Epoch 143, Loss: 0.0005566964915487915
Epoch 144, Loss: 0.0005478369741467759
Epoch 145, Loss: 0.0005377483852498699
Epoch 146, Loss: 0.00052046419295948
Epoch 147, Loss: 0.0005065247474703938
Epoch 148, Loss: 0.0004938646998198237
Epoch 149, Loss: 0.0004899770297924988
Epoch 150, Loss: 0.0004808527257409878
Epoch 151, Loss: 0.0004624811263056472
Epoch 152, Loss: 0.0004506041033891961
Epoch 153, Loss: 0.00044307033822406084
Epoch 154, Loss: 0.00044004783194395714
Epoch 155, Loss: 0.00044085795525461435
Epoch 156, Loss: 0.0004197492598905228
Epoch 157, Loss: 0.00040844430914148687
Epoch 158, Loss: 0.0004066738081746735
Epoch 159, Loss: 0.00040331827913178133
Epoch 160, Loss: 0.00038729253865312787
Epoch 161, Loss: 0.0003812388018559432
Epoch 162, Loss: 0.000376238944591023
Epoch 163, Loss: 0.00036428842940949835
Epoch 164, Loss: 0.00036204427015036346
Epoch 165, Loss: 0.00035615224187495186
Epoch 166, Loss: 0.00034750749109662137
Epoch 167, Loss: 0.00034350399655522776
Epoch 168, Loss: 0.00033509978238726034
Epoch 169, Loss: 0.00033078563428716733
Epoch 170, Loss: 0.00033096744591603055
Epoch 171, Loss: 0.0003177794707880821
Epoch 172, Loss: 0.0003141054708976299
Epoch 173, Loss: 0.0003080658367252909
Epoch 174, Loss: 0.000303906190674752
Epoch 175, Loss: 0.00029805500598740765
Epoch 176, Loss: 0.00029403659791569223
Epoch 177, Loss: 0.00029205759783508257
Epoch 178, Loss: 0.00028545283275889234
Epoch 179, Loss: 0.0002833505226590205
```

```
Epoch 180, Loss: 0.00027707008630386556
Epoch 181, Loss: 0.00027142853068653496
Epoch 182, Loss: 0.0002646974950039294
Epoch 183, Loss: 0.00026144856747123413
Epoch 184, Loss: 0.00025814678592723795
Epoch 185, Loss: 0.0002516505053790752
Epoch 186, Loss: 0.00024900555290514604
Epoch 187, Loss: 0.00024681290378794076
Epoch 188, Loss: 0.0002402110112598166
Epoch 189, Loss: 0.00023800007984391413
Epoch 190, Loss: 0.00023430451219610404
Epoch 191, Loss: 0.00023035344638628886
Epoch 192, Loss: 0.00022551573856617325
Epoch 193, Loss: 0.00022574788745259865
Epoch 194, Loss: 0.00022075892775319517
Epoch 195, Loss: 0.00021966259009786882
Epoch 196, Loss: 0.0002144397687516175
Epoch 197, Loss: 0.0002116951760399388
Epoch 198, Loss: 0.00020854149988736026
Epoch 199, Loss: 0.0002072642048005946
Epoch 200, Loss: 0.00020531284353637603
Epoch 201, Loss: 0.00020267258078092708
Epoch 202, Loss: 0.00019726043574337383
Epoch 203, Loss: 0.00019561435328796506
Epoch 204, Loss: 0.00019191373357898555
Epoch 205, Loss: 0.00018871987849706785
Epoch 206, Loss: 0.00018689024072955364
Epoch 207, Loss: 0.00018448893606546335
Epoch 208, Loss: 0.00018106851275661028
Epoch 209, Loss: 0.00017932648552232422
Epoch 210, Loss: 0.00017749061516951769
Epoch 211, Loss: 0.00017369472843711264
Epoch 212, Loss: 0.00017314601100224536
Epoch 213, Loss: 0.0001700428572803503
Epoch 214, Loss: 0.00016824936892589903
Epoch 215, Loss: 0.00016746348555898293
Epoch 216, Loss: 0.0001641390372242313
Epoch 217, Loss: 0.00016162941647053232
Epoch 218, Loss: 0.00015892263254499995
Epoch 219, Loss: 0.00015970264576026239
Epoch 220, Loss: 0.00015430597049999052
Epoch 221, Loss: 0.00015409403131343423
Epoch 222, Loss: 0.0001547547044538078
Epoch 223, Loss: 0.0001509041521785548
Epoch 224, Loss: 0.00014873682521283628
Epoch 225, Loss: 0.00014863612541375914
Epoch 226, Loss: 0.0001452707132557407
Epoch 227, Loss: 0.0001432586996088503
```

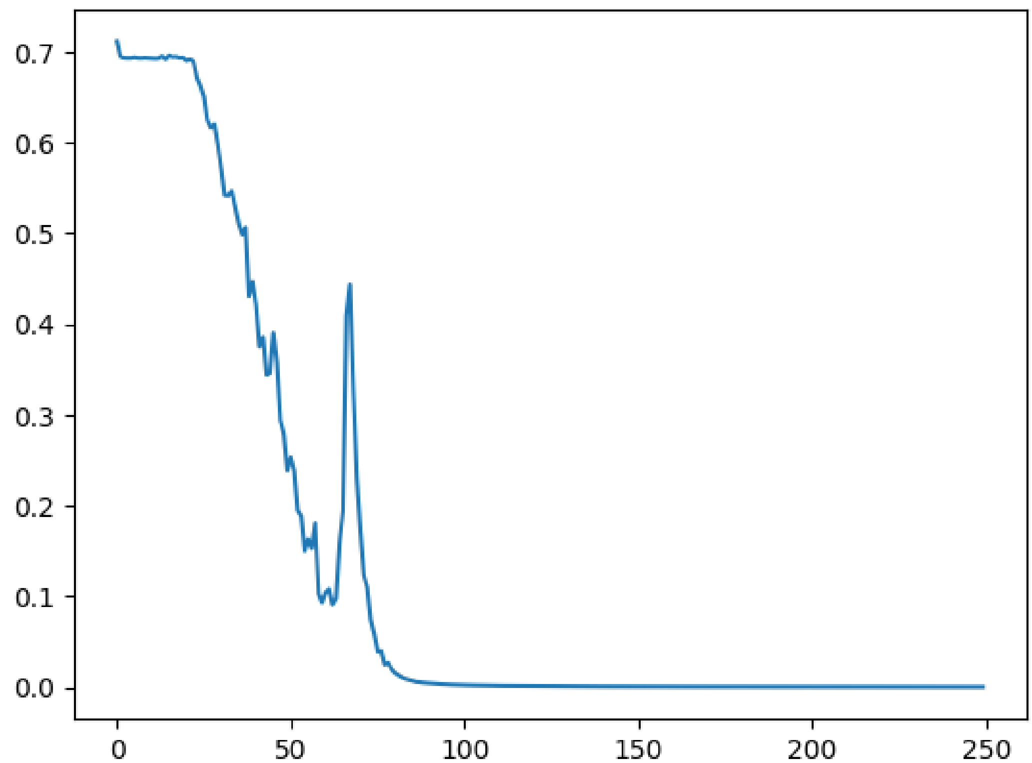
```
Epoch 228, Loss: 0.00014127274080237838
Epoch 229, Loss: 0.0001393209480738733
Epoch 230, Loss: 0.00013796852181258146
Epoch 231, Loss: 0.00013652121124323456
Epoch 232, Loss: 0.00013529791285691318
Epoch 233, Loss: 0.00013454180552798788
Epoch 234, Loss: 0.00013166007774998434
Epoch 235, Loss: 0.00012968654700671323
Epoch 236, Loss: 0.0001280343389225891
Epoch 237, Loss: 0.00012679101891990285
Epoch 238, Loss: 0.00012534707057056948
Epoch 239, Loss: 0.00012824766854464543
Epoch 240, Loss: 0.00012329532401054167
Epoch 241, Loss: 0.0001209498161188094
Epoch 242, Loss: 0.00012073973666701932
Epoch 243, Loss: 0.00011859583137265872
Epoch 244, Loss: 0.00011919803127966589
Epoch 245, Loss: 0.00011805861540779006
Epoch 246, Loss: 0.00011455695421318524
Epoch 247, Loss: 0.00011497086161398329
Epoch 248, Loss: 0.00011194554354005959
Epoch 249, Loss: 0.00011061096492994693
Epoch 250, Loss: 0.00011035511670343112
Testing Model Completed
Accuracy on the test dataset: 50.0%
```

[11]:

```
# Plotting First Training
plt.plot(trainingLosses)
```

[11]:

```
[<matplotlib.lines.Line2D at 0x1b2139219c0>]
```




[12]:
```
# Preparing the Optuna for HyperTuning Parameters

def tuning(trial):
    # Setting up the parameters
    learning_rate = trial.suggest_categorical('learning_rate', [0.1, 0.01, 0.
 ↪001, 0.0001])
    batch_size = trial.suggest_categorical('batch_size', [4, 8, 16, 32, 64] )
    num_channels = trial.suggest_categorical('num_channels', [16, 32, 64])

    # Setting up the new Proj2CNNetwork for each parameter trial
    optunaModel = proj2CNN(sizeChannel, sizeKernel, sizeStride, sizePadding,␣
 ↪sizePoolKernel, sizePoolStride, imageHeight, imageWidth)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(optunaModel.parameters(), lr = learningRate)

    # Training the model with the testing parameters
    for epoch in range(tuneEpochs):
        trainingLoss = 0.0
        for i, data in enumerate(trainLoader, 0):
            inputs, labels = data
            optimizer.zero_grad()
            outputs = optunaModel(inputs)
```

```
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            trainingLoss += loss.item()

        # Caculating the training loss from each epoch
        trainingLoss /= len(trainLoader)

    # Calculate accuracy on the validation set
    correct = 0
    total = 0
    with torch.no_grad():
        for data in testLoader:
            inputs, labels = data
            outputs = optunaModel(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
        accuracy = correct / total

    # Return the negative accuracy as Optuna tries to minimize the objective
    return -accuracy
```

[13]:
```
# Run Optuna optimization
study = optuna.create_study(direction='maximize')
study.optimize(tuning, n_trials=10)
```

```
[I 2023-11-05 07:56:04,494] A new study created in memory with name: no-
name-24a23075-005f-48c9-a31c-fe90be43f2ee
[I 2023-11-05 08:07:04,731] Trial 0 finished with value: -0.5 and parameters:
{'learning_rate': 0.0001, 'batch_size': 16, 'num_channels': 64}. Best is trial 0
with value: -0.5.
[I 2023-11-05 08:18:12,782] Trial 1 finished with value: -0.475 and parameters:
{'learning_rate': 0.0001, 'batch_size': 4, 'num_channels': 32}. Best is trial 1
with value: -0.475.
[I 2023-11-05 08:29:22,352] Trial 2 finished with value: -0.475 and parameters:
{'learning_rate': 0.1, 'batch_size': 16, 'num_channels': 16}. Best is trial 1
with value: -0.475.
[I 2023-11-05 08:40:20,132] Trial 3 finished with value: -0.475 and parameters:
{'learning_rate': 0.01, 'batch_size': 4, 'num_channels': 32}. Best is trial 1
with value: -0.475.
[I 2023-11-05 08:51:23,547] Trial 4 finished with value: -0.475 and parameters:
{'learning_rate': 0.0001, 'batch_size': 16, 'num_channels': 16}. Best is trial 1
with value: -0.475.
[I 2023-11-05 09:02:18,696] Trial 5 finished with value: -0.525 and parameters:
{'learning_rate': 0.01, 'batch_size': 4, 'num_channels': 16}. Best is trial 1
with value: -0.475.
[I 2023-11-05 09:13:52,181] Trial 6 finished with value: -0.65 and parameters:
```

```
{'learning_rate': 0.1, 'batch_size': 16, 'num_channels': 64}. Best is trial 1
with value: -0.475.
[I 2023-11-05 09:26:05,833] Trial 7 finished with value: -0.475 and parameters:
{'learning_rate': 0.01, 'batch_size': 64, 'num_channels': 32}. Best is trial 1
with value: -0.475.
[I 2023-11-05 09:39:32,624] Trial 8 finished with value: -0.525 and parameters:
{'learning_rate': 0.001, 'batch_size': 8, 'num_channels': 16}. Best is trial 1
with value: -0.475.
[I 2023-11-05 09:50:23,916] Trial 9 finished with value: -0.575 and parameters:
{'learning_rate': 0.01, 'batch_size': 64, 'num_channels': 16}. Best is trial 1
with value: -0.475.
```

[14]:

```python
# Applying Best Parameters

# Get the best hyperparameters
best_params = study.best_params

print(f"Old Learning Rate:   {learningRate}")
print(f"Old Batch Size:      {sizeBatch}")
print(f"Old Channel Size:    {sizeChannel}")

# Setting the new optimized parameters
learningRate = best_params['learning_rate']
sizeBatch = best_params['batch_size']
sizeChannel = best_params['num_channels']

print()
print(f"New Learning Rate:   {learningRate}")
print(f"New Batch Size:      {sizeBatch}")
print(f"New Channel Size:    {sizeChannel}")

# Refreshing the Model
tunedProj2CNNetwork = proj2CNN(sizeChannel, sizeKernel, sizeStride,␣
 ↪sizePadding, sizePoolKernel, sizePoolStride, imageHeight, imageWidth)
```

```
Old Learning Rate:   0.01
Old Batch Size:      16
Old Channel Size:    16

New Learning Rate:   0.0001
New Batch Size:      4
New Channel Size:    32
```

[15]:

```python
# Validating the Hypertuned Model

# Resetting Validation Loop
splitCount = 0
```

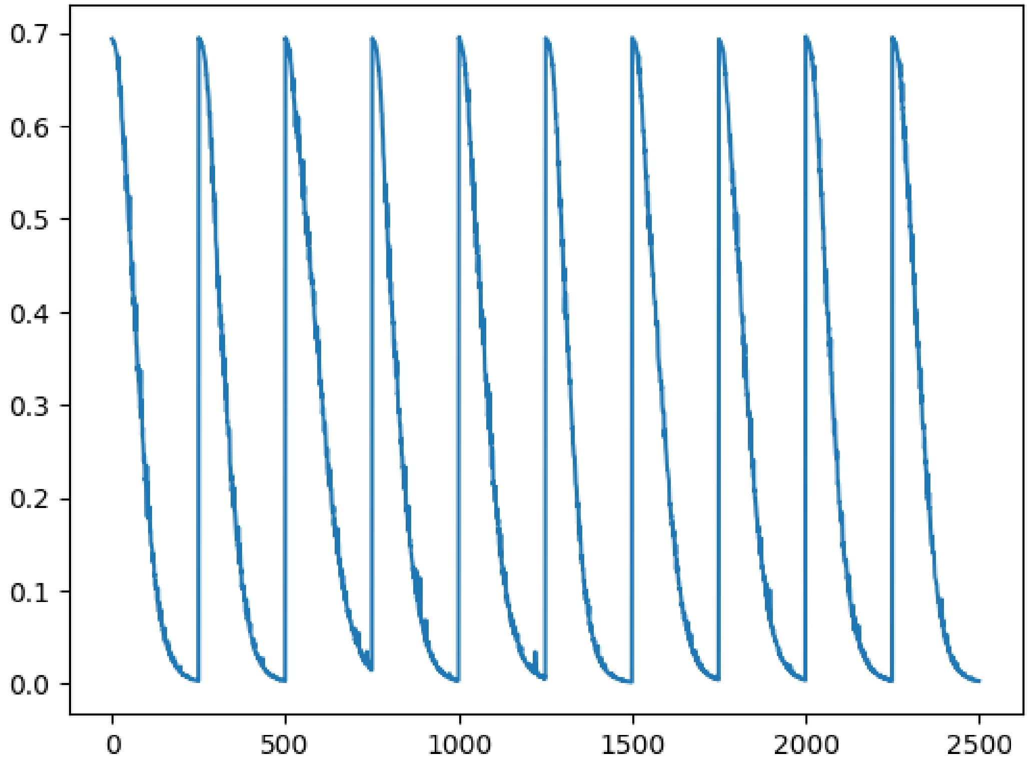
```python
# Setting up k-fold validation
cv = StratifiedKFold(n_splits = nSplits, shuffle = True, random_state = randomState)

# Defining array for recording the scores
cnn_array = []
trainingLosses = []

# Count for splits for progress tracking
splitCount = 0
# Looping through the 10 Folds
for fold, (train_index, val_index) in enumerate(cv.split(X=np.zeros(len(newDataset)), y=newDataset.labels)):
    # Reloading Datasets with tuned batch sizes
    trainLoader = torch.utils.data.DataLoader(setTrain, batch_size = sizeBatch, shuffle = True)
    testLoader = torch.utils.data.DataLoader(setTest, batch_size = sizeBatch, shuffle = True)

    # Reloading the model for each new fold
    valProj2CNNetwork = proj2CNN(sizeChannel, sizeKernel, sizeStride, sizePadding, sizePoolKernel, sizePoolStride, imageHeight, imageWidth)

    # Reloading optimizer with tuned learning rate
    optimizer = optim.Adam(valProj2CNNetwork.parameters(), lr = learningRate)

    for epoch in range(validateEpochs):
        trainingLoss = 0.0
        for i, data in enumerate(trainLoader, 0):
            inputs, labels = data
            optimizer.zero_grad()
            outputs = valProj2CNNetwork(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            trainingLoss += loss.item()
        trainingLosses.append(trainingLoss / len(trainLoader))

        # Calculating the training loss from each epoch
        trainingLoss /= len(trainLoader)

        # Begin evaluating the model
        valProj2CNNetwork.eval()
        all_predictions = []
        all_labels = []
        with torch.no_grad():
            for inputs, labels in testLoader:
```

```
                outputs = valProj2CNNetwork(inputs)
                _, predictions = torch.max(outputs, 1)
                all_predictions.extend(predictions.cpu().numpy())
                all_labels.extend(labels.cpu().numpy())
        clear_output()
        print(f"Fold {fold + 1}, Epoch {epoch + 1} / {validateEpochs}")

        # Calculate accuracy for this fold
        accuracy = accuracy_score(all_labels, all_predictions)
        cnn_array.append(accuracy)

    # Visual Confirmation of Fold Completed
    splitCount += 1
    e = datetime.datetime.now()
    print(f"Fold {fold + 1}, Accuracy: {accuracy} at {e.hour}:{e.minute}:{e.
 ↪second}")

# Final Visual Confirmation for Validation
print("Complete")
print()

# Finding metrics
print(f"The mean of 10 proj2CNNetwork models is:   {sum(cnn_array)/
 ↪len(cnn_array)}")
print(f"The stdDev of 10 proj2CNNetwork models is: {statistics.
 ↪stdev(cnn_array)}")
print(f"The 95% Confidence interval of 10 proj2CNNetwork models is: {st.t.
 ↪interval(0.95, df=len(cnn_array)-1, loc=np.mean(cnn_array), scale=st.
 ↪sem(cnn_array))}")
```

```
Fold 10, Epoch 250 / 250
Fold 10, Accuracy: 0.55 at 12:16:43
Complete

The mean of 10 proj2CNNetwork models is:   0.5597499999999962
The stdDev of 10 proj2CNNetwork models is: 0.05076138480454
The 95% Confidence interval of 10 proj2CNNetwork models is: (0.5577592260768647,
0.5617407739231353)
```

[16]:
```
# Plotting HyperTuned Training
plt.plot(trainingLosses)
```

[16]: [<matplotlib.lines.Line2D at 0x1b214110040>]




[17]:

```
# Reloading Datasets with tuned batch sizes
trainLoader = torch.utils.data.DataLoader(setTrain, batch_size = sizeBatch,␣
 ↪shuffle = True)
testLoader = torch.utils.data.DataLoader(setTest, batch_size = sizeBatch,␣
 ↪shuffle = True)

# Reloading the model for Final Model
finalProj2CNNetwork = proj2CNN(sizeChannel, sizeKernel, sizeStride,␣
 ↪sizePadding, sizePoolKernel, sizePoolStride, imageHeight, imageWidth)

# Reloading optimizer with tuned learning rate
optimizer = optim.Adam(finalProj2CNNetwork.parameters(), lr = learningRate)
2
# Resetting the TrainingLosses Tracker
trainingLosses = []

for epoch in range(finalEpochs):
    trainingLoss = 0.0
    for i, data in enumerate(trainLoader, 0):
        inputs, labels = data
        optimizer.zero_grad()
```

```
        outputs = finalProj2CNNetwork(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        trainingLoss += loss.item()
    trainingLosses.append(trainingLoss / len(trainLoader))
    print(f"Epoch {epoch+1}, Loss: {trainingLoss / len(trainLoader)}")
print("Final Model Completed")
```

```
Epoch 1, Loss: 0.6953816547989845
Epoch 2, Loss: 0.6936526790261268
Epoch 3, Loss: 0.6925975948572158
Epoch 4, Loss: 0.6929611593484879
Epoch 5, Loss: 0.692143926024437
Epoch 6, Loss: 0.692333897948265
Epoch 7, Loss: 0.6889777779579163
Epoch 8, Loss: 0.6888894617557526
Epoch 9, Loss: 0.6870663568377495
Epoch 10, Loss: 0.6870451226830483
Epoch 11, Loss: 0.6846958220005035
Epoch 12, Loss: 0.6840511351823807
Epoch 13, Loss: 0.6810416415333748
Epoch 14, Loss: 0.6792870864272118
Epoch 15, Loss: 0.6764707401394844
Epoch 16, Loss: 0.6743978172540664
Epoch 17, Loss: 0.6699081733822823
Epoch 18, Loss: 0.6653459146618843
Epoch 19, Loss: 0.6600169345736504
Epoch 20, Loss: 0.6565339356660843
Epoch 21, Loss: 0.6492408573627472
Epoch 22, Loss: 0.6521514996886253
Epoch 23, Loss: 0.6417554646730423
Epoch 24, Loss: 0.6315587967634201
Epoch 25, Loss: 0.6306078866124153
Epoch 26, Loss: 0.6188861653208733
Epoch 27, Loss: 0.6148751638829708
Epoch 28, Loss: 0.6255301177501679
Epoch 29, Loss: 0.6015413336455822
Epoch 30, Loss: 0.5988882511854172
Epoch 31, Loss: 0.591367669403553
Epoch 32, Loss: 0.5830299191176891
Epoch 33, Loss: 0.5772470600903035
Epoch 34, Loss: 0.5734493486583233
Epoch 35, Loss: 0.5594895794987679
Epoch 36, Loss: 0.5868122965097428
Epoch 37, Loss: 0.54984095916152
Epoch 38, Loss: 0.5378112852573395
Epoch 39, Loss: 0.5341049388051033
```

```
Epoch 40, Loss: 0.5639385603368282
Epoch 41, Loss: 0.5213572531938553
Epoch 42, Loss: 0.5126014284789562
Epoch 43, Loss: 0.5116907253861427
Epoch 44, Loss: 0.4997977539896965
Epoch 45, Loss: 0.5044722374528646
Epoch 46, Loss: 0.4828867293894291
Epoch 47, Loss: 0.4751699589192867
Epoch 48, Loss: 0.46017961464822293
Epoch 49, Loss: 0.46073328480124476
Epoch 50, Loss: 0.4523685425519943
Epoch 51, Loss: 0.45271528176963327
Epoch 52, Loss: 0.4373596772551537
Epoch 53, Loss: 0.4361647550016642
Epoch 54, Loss: 0.42154717855155466
Epoch 55, Loss: 0.4186294261366129
Epoch 56, Loss: 0.41083934493362906
Epoch 57, Loss: 0.41760296318680046
Epoch 58, Loss: 0.4037754122167826
Epoch 59, Loss: 0.3905705217272043
Epoch 60, Loss: 0.38704228326678275
Epoch 61, Loss: 0.3815606884658337
Epoch 62, Loss: 0.4001576840877533
Epoch 63, Loss: 0.38840155899524687
Epoch 64, Loss: 0.3591211372986436
Epoch 65, Loss: 0.36695106960833074
Epoch 66, Loss: 0.35382183343172074
Epoch 67, Loss: 0.3637528885155916
Epoch 68, Loss: 0.3304796241223812
Epoch 69, Loss: 0.3402321795001626
Epoch 70, Loss: 0.3353203976526856
Epoch 71, Loss: 0.3193668872117996
Epoch 72, Loss: 0.3219599362462759
Epoch 73, Loss: 0.3006631413474679
Epoch 74, Loss: 0.3005931143648922
Epoch 75, Loss: 0.3015873868018389
Epoch 76, Loss: 0.30624825935810807
Epoch 77, Loss: 0.28803330650553105
Epoch 78, Loss: 0.289490656927228
Epoch 79, Loss: 0.27013730593025687
Epoch 80, Loss: 0.26465811785310506
Epoch 81, Loss: 0.26022214014083145
Epoch 82, Loss: 0.2514844586607069
Epoch 83, Loss: 0.2505134640261531
Epoch 84, Loss: 0.2450786491855979
Epoch 85, Loss: 0.2372743481770158
Epoch 86, Loss: 0.23039869079366326
Epoch 87, Loss: 0.22809312604367732
```

```
Epoch 88, Loss: 0.22307671718299388
Epoch 89, Loss: 0.21636084076017142
Epoch 90, Loss: 0.22145562414079906
Epoch 91, Loss: 0.21118699198123067
Epoch 92, Loss: 0.20074317194521427
Epoch 93, Loss: 0.19976813169196247
Epoch 94, Loss: 0.1958167316392064
Epoch 95, Loss: 0.19216573159210384
Epoch 96, Loss: 0.1888901896774769
Epoch 97, Loss: 0.1778238764964044
Epoch 98, Loss: 0.17531251041218637
Epoch 99, Loss: 0.16879257978871465
Epoch 100, Loss: 0.1931649715639651
Epoch 101, Loss: 0.1594525713007897
Epoch 102, Loss: 0.1578191528096795
Epoch 103, Loss: 0.15911199804395437
Epoch 104, Loss: 0.15115397023037075
Epoch 105, Loss: 0.14861198081634938
Epoch 106, Loss: 0.1457661590538919
Epoch 107, Loss: 0.139190472336486
Epoch 108, Loss: 0.14093920988962055
Epoch 109, Loss: 0.13040232276543975
Epoch 110, Loss: 0.12622534562833607
Epoch 111, Loss: 0.1342622465454042
Epoch 112, Loss: 0.1271848704200238
Epoch 113, Loss: 0.1183203861117363
Epoch 114, Loss: 0.11657453801017255
Epoch 115, Loss: 0.11148442164994776
Epoch 116, Loss: 0.11059515313245356
Epoch 117, Loss: 0.10544542900752277
Epoch 118, Loss: 0.10078507072757929
Epoch 119, Loss: 0.09855325943790376
Epoch 120, Loss: 0.09890603316016496
Epoch 121, Loss: 0.10178144661476836
Epoch 122, Loss: 0.09908081712201237
Epoch 123, Loss: 0.08896644678898155
Epoch 124, Loss: 0.10973862253595143
Epoch 125, Loss: 0.08700500861741603
Epoch 126, Loss: 0.08240571208298206
Epoch 127, Loss: 0.08137696560006588
Epoch 128, Loss: 0.0879405050072819
Epoch 129, Loss: 0.07697583194822073
Epoch 130, Loss: 0.07283225272549317
Epoch 131, Loss: 0.07048812576103955
Epoch 132, Loss: 0.0743758428376168
Epoch 133, Loss: 0.06826027287170292
Epoch 134, Loss: 0.06288090217858552
Epoch 135, Loss: 0.06483187112025916
```

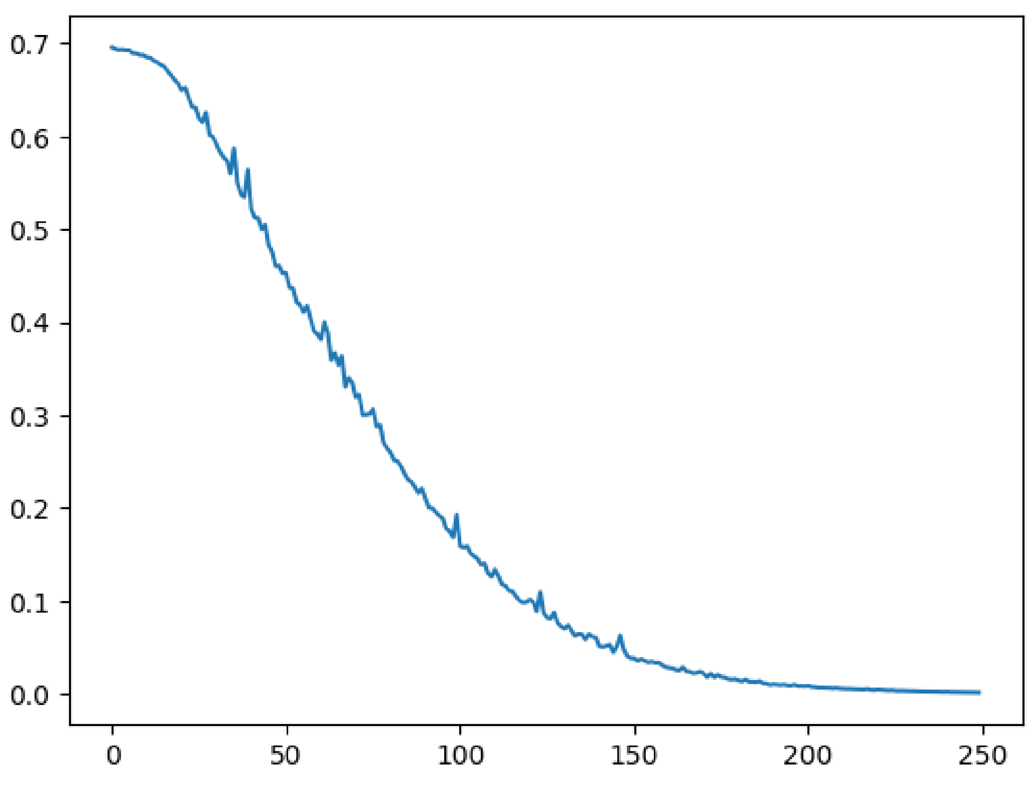
```
Epoch 136, Loss: 0.0645285170758143
Epoch 137, Loss: 0.0589117971714586
Epoch 138, Loss: 0.06465839112643153
Epoch 139, Loss: 0.06173241826472804
Epoch 140, Loss: 0.06115853349911049
Epoch 141, Loss: 0.05161747686797753
Epoch 142, Loss: 0.05083039625314996
Epoch 143, Loss: 0.05212299434933811
Epoch 144, Loss: 0.0532821454340592
Epoch 145, Loss: 0.04501801144797355
Epoch 146, Loss: 0.051606938149780034
Epoch 147, Loss: 0.06335436024237424
Epoch 148, Loss: 0.0479138758732006
Epoch 149, Loss: 0.041010948951588945
Epoch 150, Loss: 0.03890181967290118
Epoch 151, Loss: 0.03868246026104316
Epoch 152, Loss: 0.03589554104255512
Epoch 153, Loss: 0.037790627725189554
Epoch 154, Loss: 0.03602766768308356
Epoch 155, Loss: 0.03431098682340235
Epoch 156, Loss: 0.03516790361900348
Epoch 157, Loss: 0.033819003199459984
Epoch 158, Loss: 0.03416188210248947
Epoch 159, Loss: 0.03132801137981005
Epoch 160, Loss: 0.029078527953242884
Epoch 161, Loss: 0.02837052013492212
Epoch 162, Loss: 0.027835647756000982
Epoch 163, Loss: 0.026208246091846375
Epoch 164, Loss: 0.025572060950798912
Epoch 165, Loss: 0.029046917776577176
Epoch 166, Loss: 0.02462889568996616
Epoch 167, Loss: 0.024124838334682864
Epoch 168, Loss: 0.022489580151159316
Epoch 169, Loss: 0.022995400575746318
Epoch 170, Loss: 0.02434007810370531
Epoch 171, Loss: 0.021930218080524356
Epoch 172, Loss: 0.018334102192602585
Epoch 173, Loss: 0.022271136357448994
Epoch 174, Loss: 0.018120792679837905
Epoch 175, Loss: 0.020848958694841713
Epoch 176, Loss: 0.018590826549916527
Epoch 177, Loss: 0.017946824035607278
Epoch 178, Loss: 0.016345329614705407
Epoch 179, Loss: 0.0158302505093161
Epoch 180, Loss: 0.016218167392071336
Epoch 181, Loss: 0.014736974129846203
Epoch 182, Loss: 0.013820832512283231
Epoch 183, Loss: 0.015850767577649093
```

```
Epoch 184, Loss: 0.013398599723586813
Epoch 185, Loss: 0.013461280980845914
Epoch 186, Loss: 0.013006302216672339
Epoch 187, Loss: 0.014066106779500842
Epoch 188, Loss: 0.011576816460001282
Epoch 189, Loss: 0.01146032997930888
Epoch 190, Loss: 0.009928345096705015
Epoch 191, Loss: 0.01067964479298098
Epoch 192, Loss: 0.010199709232256281
Epoch 193, Loss: 0.009842379970359616
Epoch 194, Loss: 0.010310667182784528
Epoch 195, Loss: 0.009425789960369002
Epoch 196, Loss: 0.009037462138803676
Epoch 197, Loss: 0.010313289623445599
Epoch 198, Loss: 0.00872109078190988
Epoch 199, Loss: 0.008844670376856812
Epoch 200, Loss: 0.008584909270575735
Epoch 201, Loss: 0.009060910728294402
Epoch 202, Loss: 0.007805258937878534
Epoch 203, Loss: 0.007584179294644855
Epoch 204, Loss: 0.006861704319453566
Epoch 205, Loss: 0.0070331087728845885
Epoch 206, Loss: 0.006672055251692654
Epoch 207, Loss: 0.006745259724266361
Epoch 208, Loss: 0.0062670423387316985
Epoch 209, Loss: 0.0066623661339690445
Epoch 210, Loss: 0.006132720566529315
Epoch 211, Loss: 0.005867850057256874
Epoch 212, Loss: 0.0055952016373339575
Epoch 213, Loss: 0.005705759640113684
Epoch 214, Loss: 0.005391674682687153
Epoch 215, Loss: 0.0053336424629378595
Epoch 216, Loss: 0.005104608291730983
Epoch 217, Loss: 0.004989290356752463
Epoch 218, Loss: 0.005574586546572391
Epoch 219, Loss: 0.004626627683319384
Epoch 220, Loss: 0.0045191477271146144
Epoch 221, Loss: 0.004960669785214122
Epoch 222, Loss: 0.004582827433478087
Epoch 223, Loss: 0.004222066386137158
Epoch 224, Loss: 0.004015908206929453
Epoch 225, Loss: 0.00432181974319974
Epoch 226, Loss: 0.003791714469116414
Epoch 227, Loss: 0.003563238291098969
Epoch 228, Loss: 0.003727631781657692
Epoch 229, Loss: 0.003463277684932109
Epoch 230, Loss: 0.0033349871322570835
Epoch 231, Loss: 0.0033392280820407905
```

```
Epoch 232, Loss: 0.003092093991290312
Epoch 233, Loss: 0.0029946117210783996
Epoch 234, Loss: 0.002965033766668057
Epoch 235, Loss: 0.0029015973865170962
Epoch 236, Loss: 0.002716728144150693
Epoch 237, Loss: 0.002916722277586814
Epoch 238, Loss: 0.002739195403410122
Epoch 239, Loss: 0.002634281165956054
Epoch 240, Loss: 0.0024515600813174387
Epoch 241, Loss: 0.002651785452326294
Epoch 242, Loss: 0.0023398108587571187
Epoch 243, Loss: 0.0023838655666228
Epoch 244, Loss: 0.0022449047064583283
Epoch 245, Loss: 0.0021515148501748626
Epoch 246, Loss: 0.002061366647649265
Epoch 247, Loss: 0.0020464717104914597
Epoch 248, Loss: 0.0019929654104998916
Epoch 249, Loss: 0.0019332213189045433
Epoch 250, Loss: 0.0018948803917737678
Final Model Completed
```

[18]:
```
# Plotting Final Training
plt.plot(trainingLosses)
```

[18]: [<matplotlib.lines.Line2D at 0x1b21416ee00>]



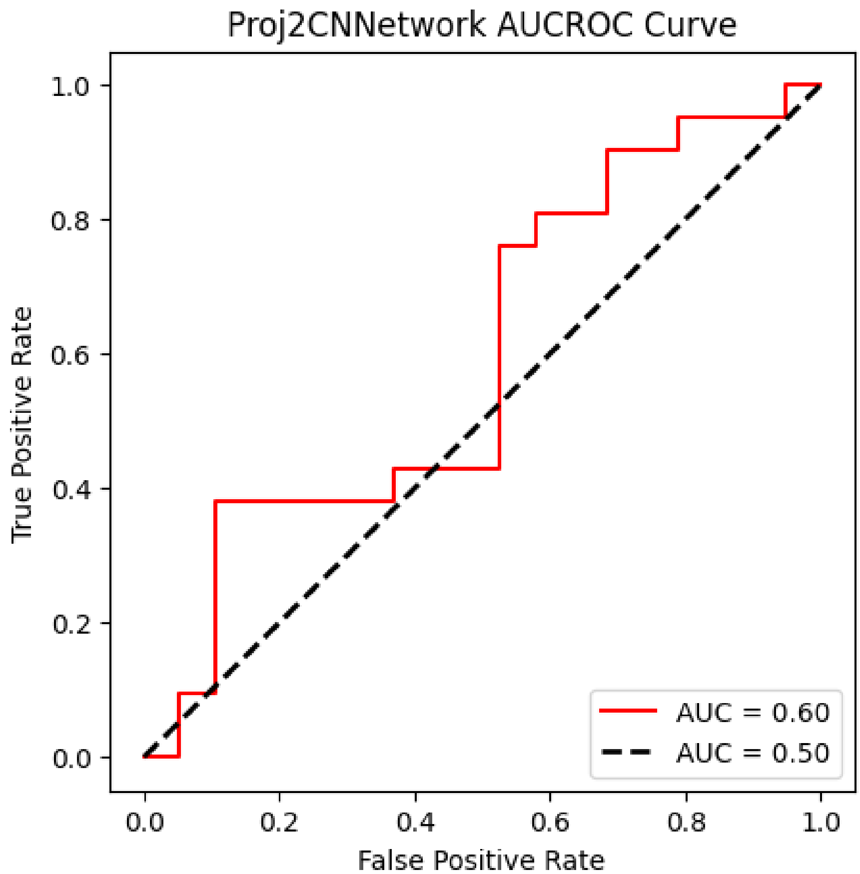
[19]:

```
# # Resetting variables for validation and creating future metrics
correct = 0
total = 0
allPredictions = []
allLabels = []

# Validating the tested model
with torch.no_grad():
    for data in testLoader:
        inputs, labels = data
        outputs = finalProj2CNNetwork(inputs)

        # Applying softmax activation to get probabilities
        probabilities = F.softmax(outputs, dim=1)

        _, predicted = torch.max(probabilities, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

        # Making a list of all predictions and labels
        allPredictions.extend(probabilities.cpu().numpy()[:, 1])
        allLabels.extend(labels.cpu().numpy())

print(f"Accuracy on the test dataset: {100 * correct / total}%")
```

```
Accuracy on the test dataset: 52.5%
```

[20]:

```
# Suitable Metric 01

# Calculate ROC curve
fpr, tpr, _ = roc_curve(allLabels, allPredictions)

# Calculate AUC-ROC
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(5, 5))
plt.plot(fpr, tpr, label = f'AUC = {roc_auc:.2f}', color = 'red')
plt.plot([0, 1], [0, 1], lw=2, label = "AUC = 0.50", linestyle='--',␣
 ↪color='black')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('Proj2CNNetwork AUCROC Curve')
plt.legend(loc='lower right')
plt.show()
```

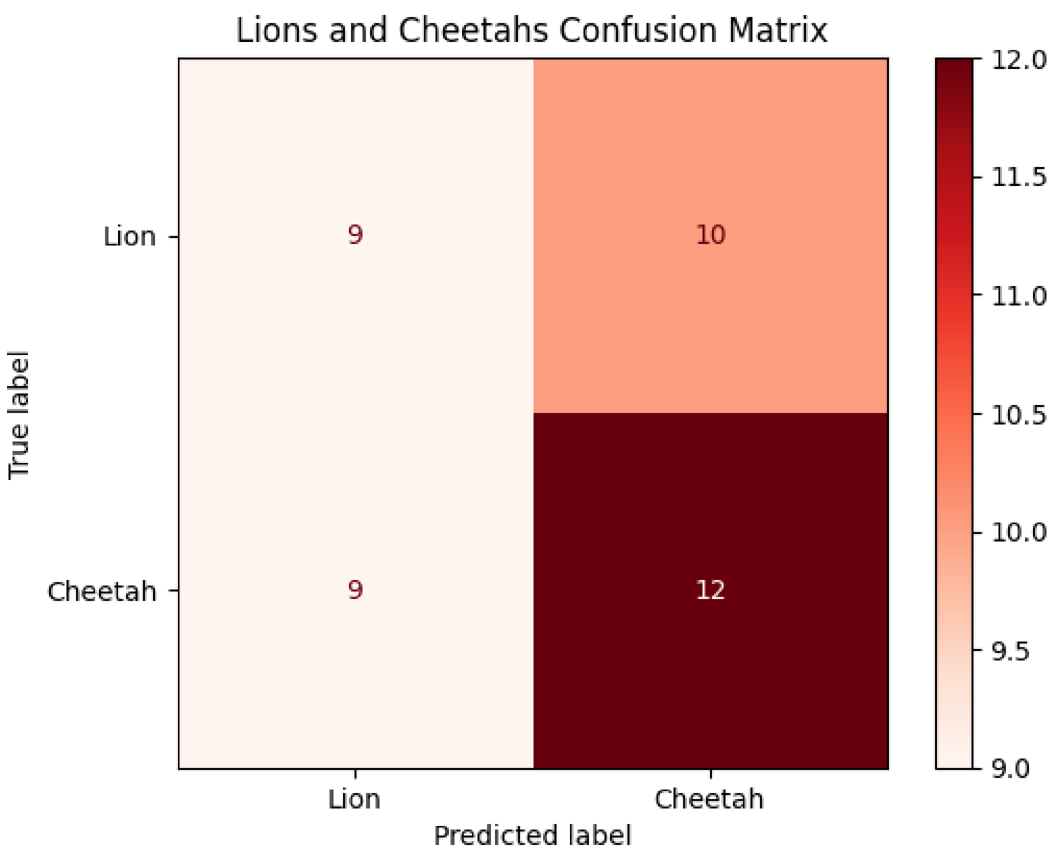



[21]:
```
# # Resetting variables for validation and creating future metrics
correct = 0
total = 0
allPredictions = []
allLabels = []

# Validating the tested model
with torch.no_grad():
    for data in testLoader:
        inputs, labels = data
        outputs = finalProj2CNNetwork(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

        # Making a list of all predictions and labels
        allPredictions.extend(predicted.cpu().numpy())
        allLabels.extend(labels.cpu().numpy())
```

```python
print(f"Accuracy on the test dataset: {100 * correct / total}%")
```

```
Accuracy on the test dataset: 52.5%
```

[22]:
```python
# Suitable Metric 02

# Making the Matrix
conMatrix = confusion_matrix(allLabels, allPredictions)

# Formatting the Matrix
disp = ConfusionMatrixDisplay(confusion_matrix=conMatrix,
                              display_labels=['Lion', 'Cheetah'])
disp.plot(cmap='Reds', values_format='d')
plt.title('Lions and Cheetahs Confusion Matrix')

# Displaying the Matrix
plt.show()
```




[23]:
```python
# Saving the model for the Webpage
# torch.save(proj2CNNetwork.state_dict(),".\\Website\\Proj2.py")
```

```
torch.save(finalProj2CNNetwork.state_dict(),".\\Website\\Proj2BIG.py")
print("Model Saved Succesfully")
```

```
Model Saved Succesfully
```

[ ]:

```

```