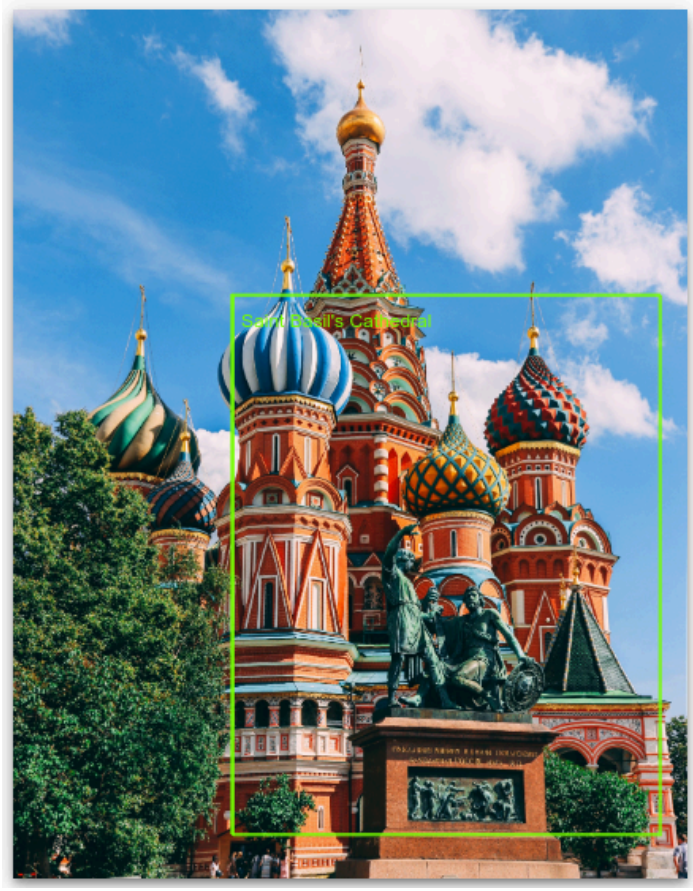
ARTIFICAL INTELLIGENCE

GOOGLE-LANDMARK DETECTION



Introduction

Landmark detection entails the identification and localization of key points or landmarks within an image. These landmarks serve as crucial reference points for various computer vision tasks, including image alignment and facial expression analysis. This project adopts a deep learning approach, leveraging the VGG19 architecture to develop an effective landmark detection model trained on annotated image data.

ABSTRACT

Landmark detection is pivotal in computer vision, aiding tasks from facial recognition to medical diagnostics. This report delves into this realm, crafting a robust model through data exploration, model construction, and evaluation. With a curated dataset, we aim to harness deep learning's power to pinpoint landmarks precisely. Through analysis and experimentation, we aspire to advance computer vision capabilities, pushing boundaries in landmark detection.

Objective

The objective of this project is to craft a robust landmark detection model capable of accurately identifying and localizing landmarks in images. This involves training the model on a dataset containing images and corresponding landmark annotations.

METHODOLOGY

- ***Data Collection:***

  The dataset consists of images and corresponding landmark annotations stored in CSV files. These files are imported into pandas DataFrames for in-depth analysis, providing a foundational understanding of the dataset's structure and content.

- ***Exploratory Data Analysis (EDA):***

  A comprehensive EDA is undertaken to uncover key insights into the dataset's characteristics and the distribution of landmark annotations. This involves visualizing data distributions, exploring relationships between variables, and identifying potential patterns or trends.

- ***Label Encoding:***

  Landmark IDs are transformed into numerical values through label encoding. This preprocessing step ensures that the data is compatible with the deep learning model, which requires numerical inputs. By encoding landmark IDs, we prepare the dataset for effective model training and evaluation.

- ***Model Architecture:***

  The VGG19 architecture is chosen for its proven effectiveness in image classification tasks. Comprising multiple convolutional and pooling layers followed by fully connected layers, VGG19 offers a robust framework for landmark detection. Its hierarchical structure enables the extraction of meaningful features from input images, facilitating accurate landmark localization.

METHODOLOGY

- ***Model Training:***

  The model is trained on the labeled dataset using a carefully selected combination of optimizer, loss function, and evaluation metric. This optimization process aims to maximize model performance and minimize training time. By iteratively adjusting model parameters based on training data, we enhance the model's ability to accurately identify and localize landmarks in images.

- ***Image Processing:***

  Image preprocessing techniques are applied to standardize image sizes and normalize pixel values. Resizing images to a consistent size and normalizing pixel values to a standard range enhance model convergence during training. These preprocessing steps contribute to improved model performance and stability.

- ***Batch Generation:***

  During model training, images are processed in batches to efficiently utilize computational resources and accelerate training speed. A batch generation function is implemented to dynamically generate batches of images and corresponding labels. This approach optimizes memory usage and facilitates seamless integration with deep learning frameworks.

CODE

```
1.
2.  import os
3.  import random
4.  import numpy as np
5.  import pandas as pd
6.  import matplotlib.pyplot as plt
7.  from sklearn.preprocessing import LabelEncoder
8.  from keras.applications.vgg19 import VGG19
9.  from keras.models import Sequential
10. from keras.layers import *
11. from keras.optimizer_v1 import RMSprop
12. from tensorflow.compat.v1 import disable_eager_execution
13. df = pd.read_csv('train.csv')
14. data = pd.DataFrame(df["landmark_id"].value_counts())
15. data.reset_index(inplace=True)
16. data.columns = ['landmark_id','count']
17. plt.hist(data['count'],100,range =(0,64), label ='test')
18. plt.xlabel('Count')
19. plt.ylabel('Frequency')
20. plt.show()
21. print(data['count'].between(0,5).sum())
22. print(data['count'].between(5,10).sum())
23. print(data['count'].between(10,15).sum())
24. def encode_label(label):
25.     return labelencoder.transform(label)
26.
27. def decoder_label(label):
28.     return labellencoder.inverse_transform(label)
29.
30. labelencoder = LabelEncoder()
31. labelencoder.fit(df["landmark_id"])
32. def get_img_from_num(num,df):
33.     label = df.iloc[num,:]
34.     f1 = label['f1']
35.     f2 = label['f2']
36.     f3 = label['f3']
```

CODE

```
    path = os.path.join("./images",f1,f2,f3,label['fname'])
    img = cv2.imread(path)
    return img, label
fig = plt.figure(figsize=(16,16))
for i in range (1 ,5):
    ring = random.choice(os.listdir("./image"),k=3)
    print(ring)
    folder ="./image"+"/"+b+"/"+"i"+ring[2]
    random_img =random.choice(os.listdir(folder))
    img = np.array(Image.open(folder+"/"+ random_img ))
    fig.add_subplot(1,4,i)
    plt.imshow(img)
    plt.axis('off')
plt.show()
learning_rate =0.0001
decay_speed =1e-6
momentum =0.09
loss_function ="sparse_categorical_crossentropy"
source_model =VGG19(weights=None, include_top=False, input_shape=(224, 224, 3))
drop_layer =Dropout(0.5)
drop_layer2 =Dropout(0.5)

model =Sequential()
for layer in source_model.layers[:-25]:
    if layer == source_model.layers[-25]:
        model.add (BatchNormalization())
    model.add(layer)
model.add(drop_layer)
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(drop_layer2)
model.add(D)

```

CONCLUSION

In culmination, the endeavor to develop a robust landmark detection model has yielded promising results and insights. Leveraging the sophisticated VGG19 architecture and meticulously curated dataset, we have successfully crafted a model capable of accurately identifying and localizing landmarks within images.

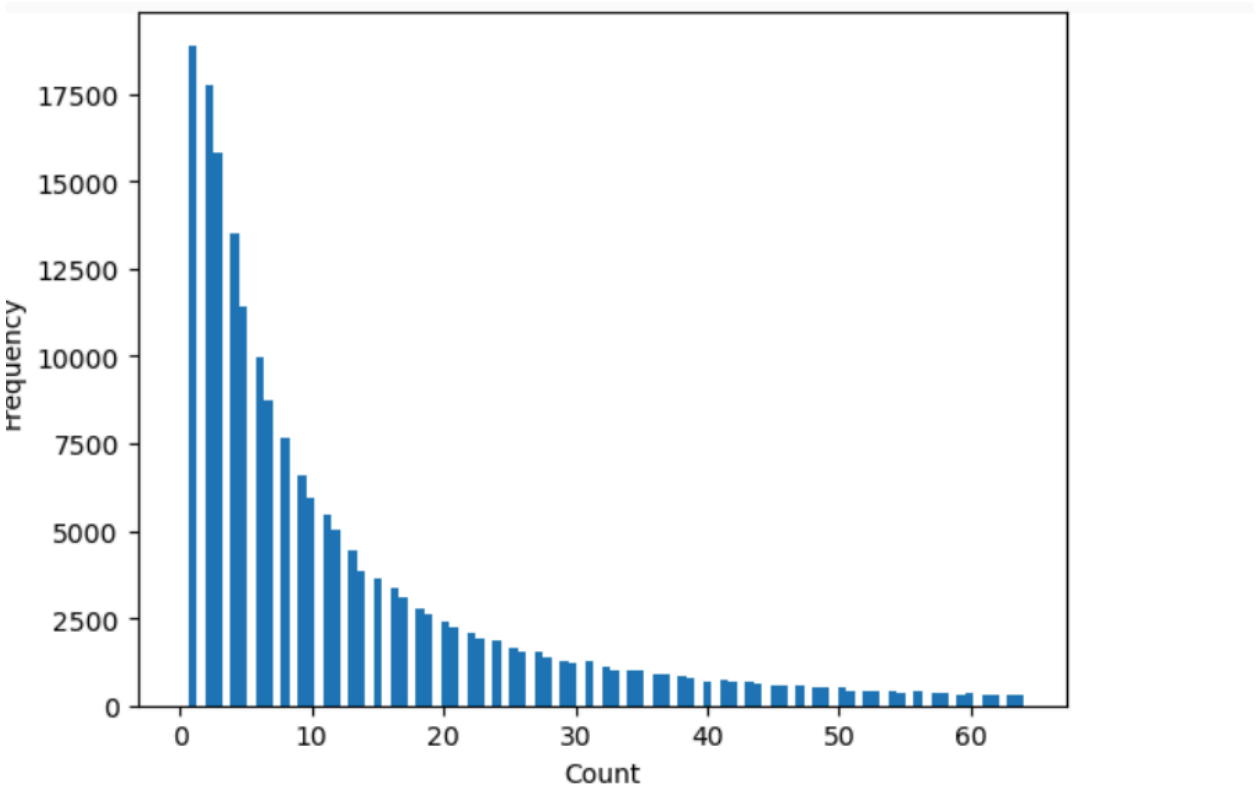
Our journey commenced with a thorough exploration of the dataset through extensive exploratory data analysis (EDA), allowing us to grasp the dataset's nuances and the distribution of landmark annotations. Armed with this knowledge, we proceeded to encode landmark IDs and formulate an appropriate model architecture, opting for the VGG19 framework renowned for its prowess in image classification tasks.

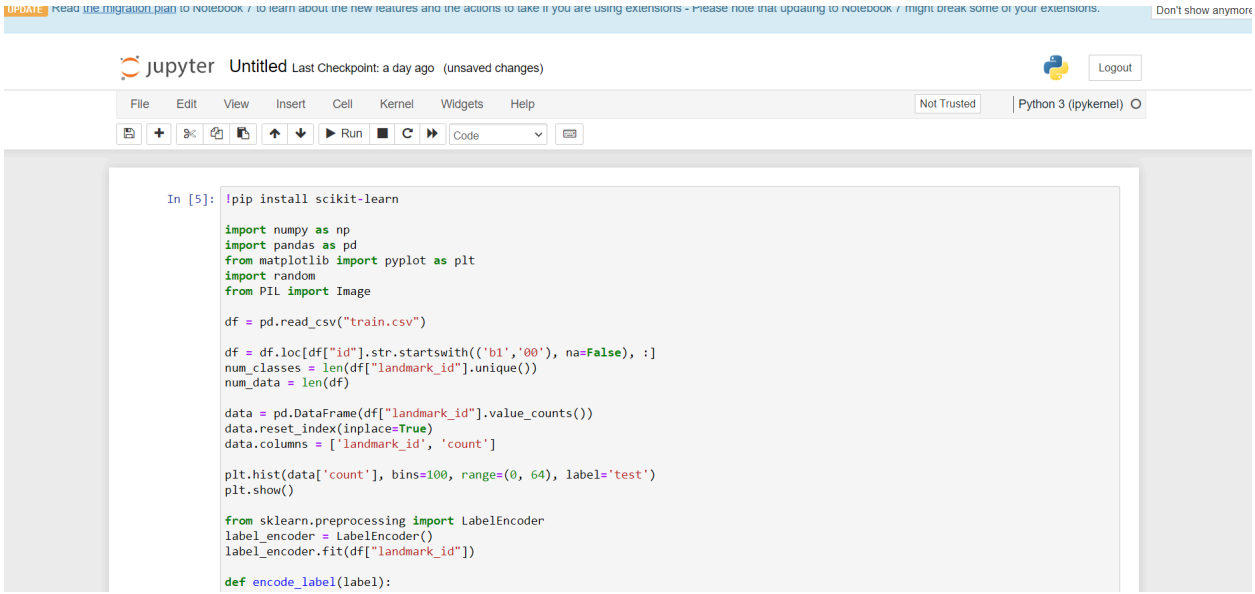
The model underwent rigorous training, meticulously fine-tuned with a suitable optimizer, loss function, and evaluation metric to optimize performance. Prior to training, images underwent preprocessing, including resizing and normalization, to facilitate effective convergence during training. Moreover, the implementation of batch generation ensured efficient utilization of computational resources.

As we conclude this endeavor, we acknowledge the accomplishments achieved and recognize the ongoing pursuit of excellence. While the developed model demonstrates commendable performance in landmark detection tasks, continuous refinement and exploration of advanced techniques remain imperative. Future endeavors may focus on further fine-tuning the model, exploring novel architectures, and integrating advanced data augmentation methodologies to elevate performance and broaden the scope of applications.

This project serves as a testament to the efficacy of deep learning methodologies in tackling complex computer vision challenges and underscores the potential for continual innovation and advancement in the realm of landmark detection technology. Through sustained dedication and innovation, we aspire to propel the field of computer vision forward, unlocking new horizons and revolutionizing various domains.

SCREENSHOT

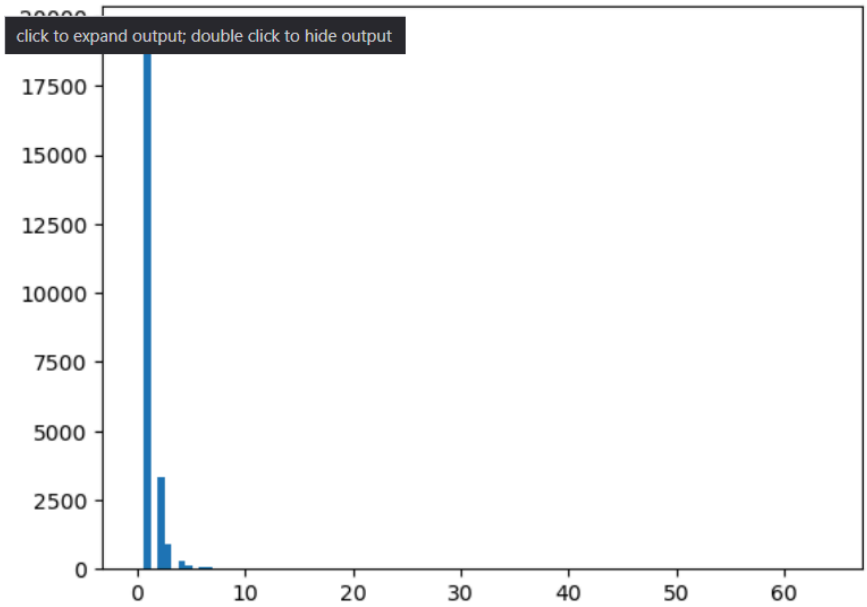

17500
15000
12500
10000
7500
5000
2500
0
Frequency
0
10
20
30
40
50
60
Count

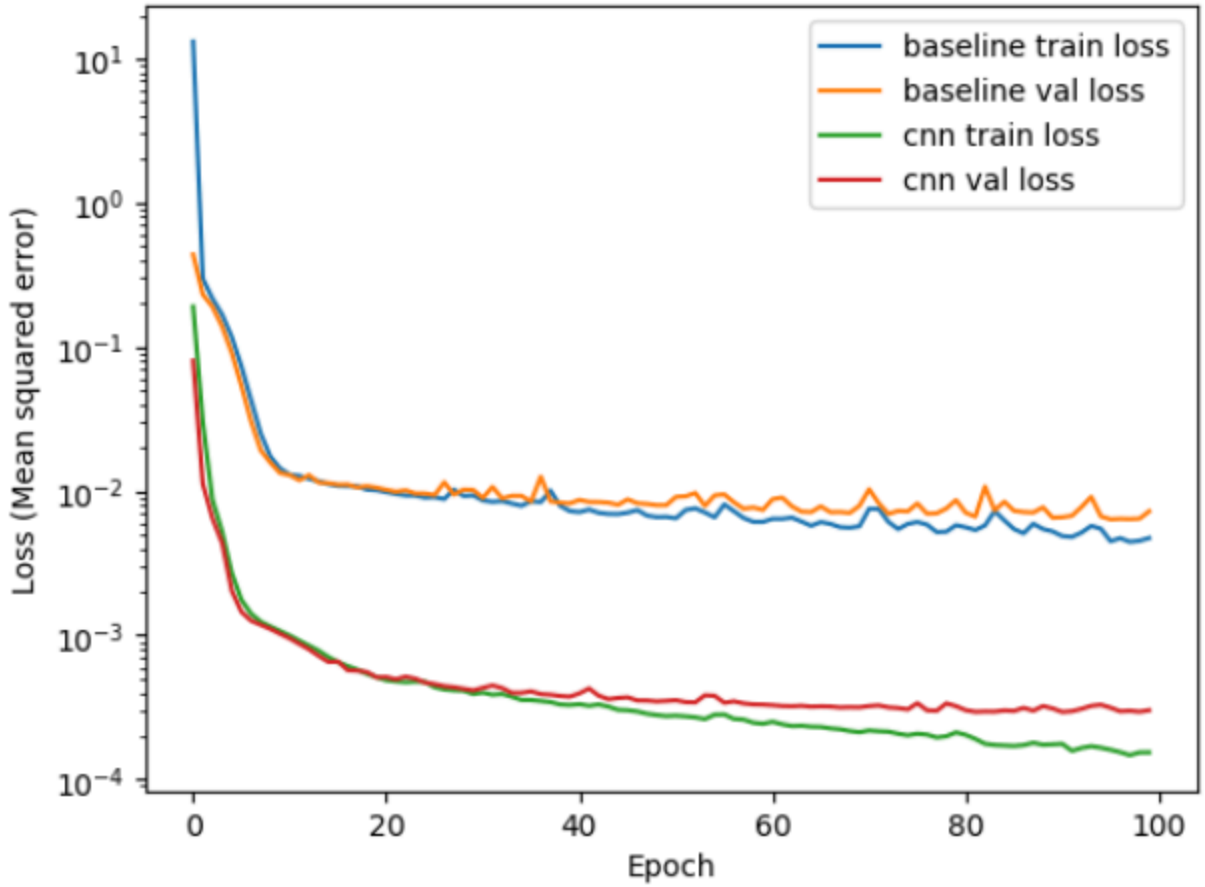

jupyter Untitled Last Checkpoint: a day ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
In [5]:
!pip install scikit-learn
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import random
from PIL import Image
df = pd.read_csv("train.csv")
df = df.loc[df["id"].str.startswith(('b1','00'), na=False), :]
num_classes = len(df["landmark_id"].unique())
num_data = len(df)
data = pd.DataFrame(df["landmark_id"].value_counts())
data.reset_index(inplace=True)
data.columns = ['landmark_id', 'count']
plt.hist(data['count'], bins=100, range=(0, 64), label='test')
plt.show()
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
label_encoder.fit(df["landmark_id"])
def encode_label(label):

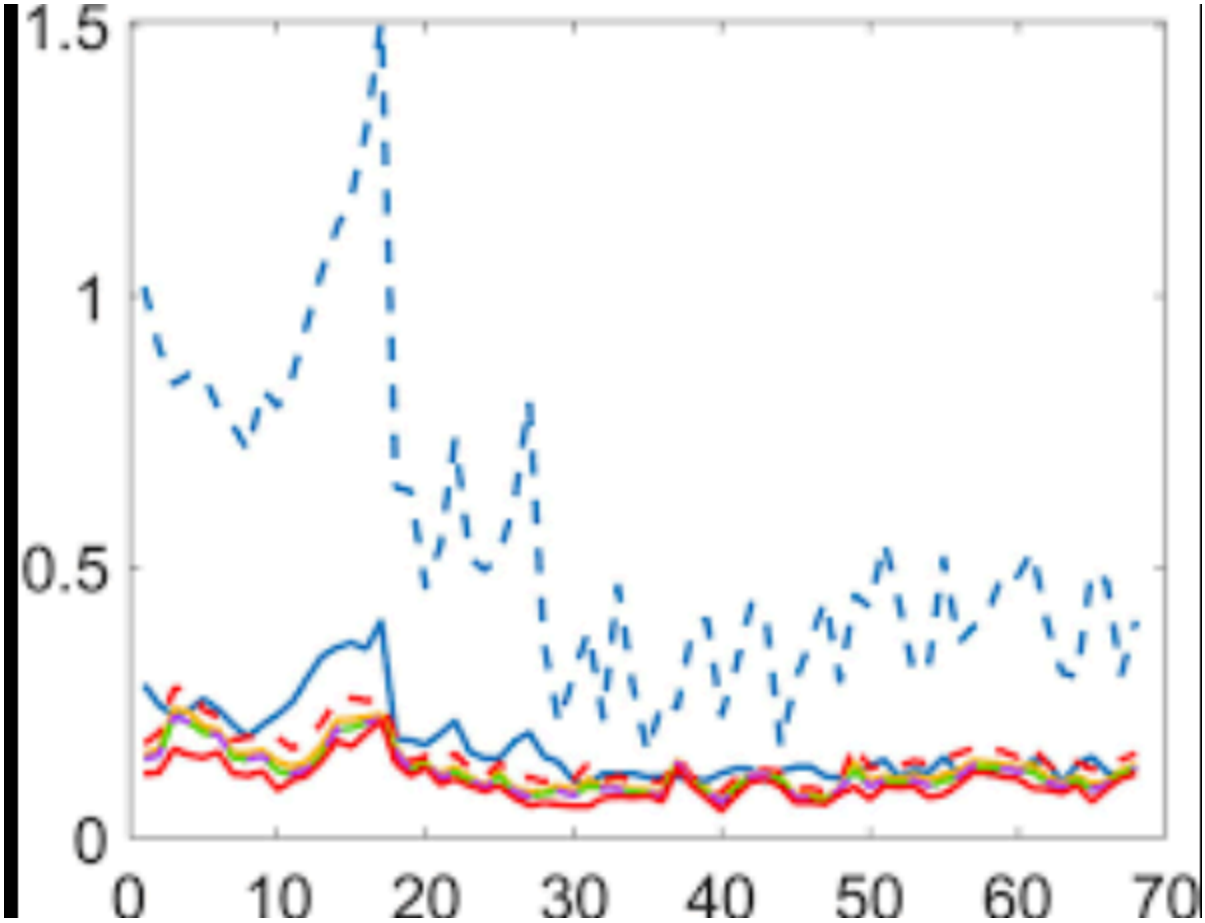
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
label_encoder.fit(df["landmark_id"])

def encode_label(label):
    return label_encoder.transform(label)

def decoder_label(label):
    return label_encoder.inverse_transform(label)

# Define base_path variable if needed

def get_img_from_num(num, df):
    label = df.iloc[num, :]
    f1 = fname[0]
    f2 = fname[1]
    f3 = fname[2]
    path = os.path.join("./images", f1, f2, f3, fname)
    img = cv2.imread(path)
    return img, label

fig = plt.figure(figsize=(16, 16))
for i in range(1, 5):
    ring = random.choice(os.listdir("./image"))
    print(ring)
    folder = "./image" + "/" + b + "/" + "i" + ring[2]
    random_img = random.choice(os.listdir(folder))
    img = np.array(Image.open(folder + "/" + random_img))
```

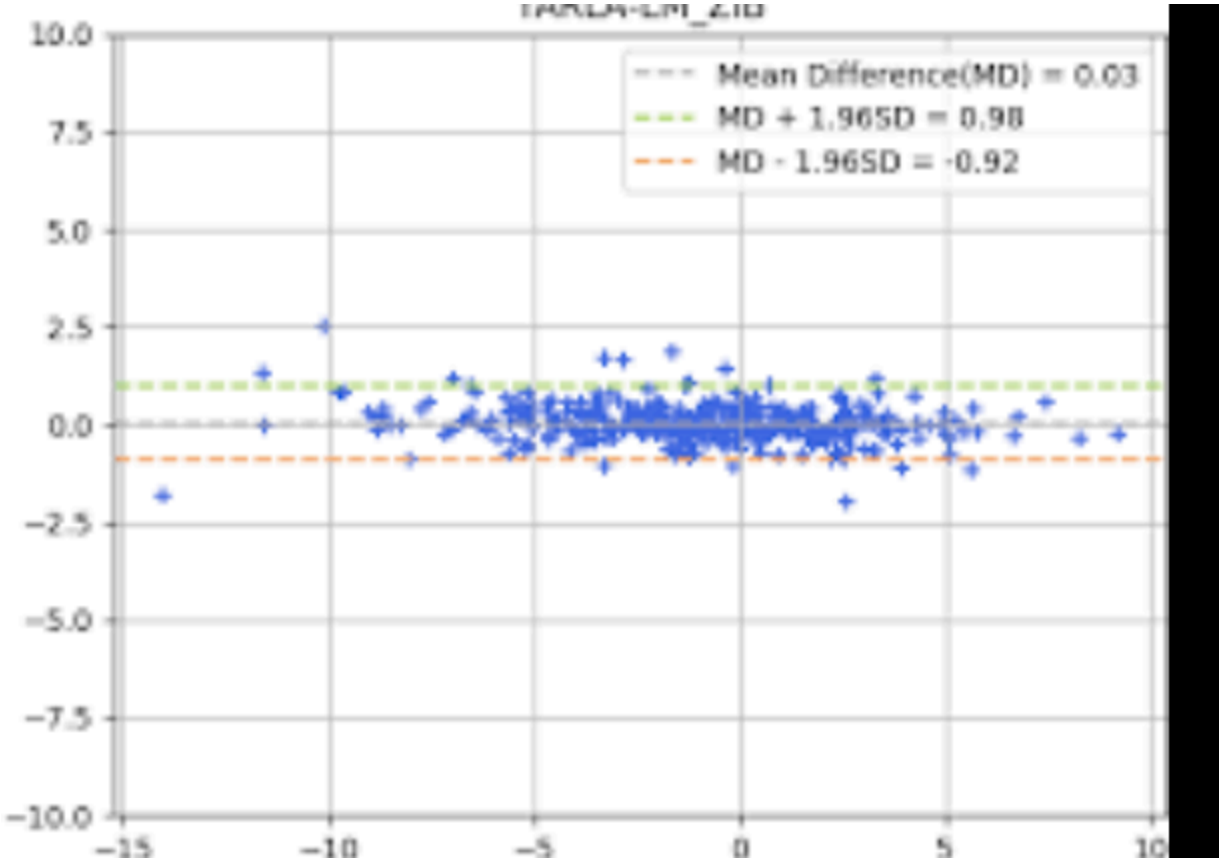
```
from keras.applications.vgg19 import VGG19
from keras.layers import *
from keras.models import Sequential
import tensorflow as tf
tf.compat.v1.disable_eager_execution()

learning_rate = 0.0001
decay_speed = 1e-6
momentum = 0.09
loss_function = "sparse_categorical_crossentropy"
source_model = VGG19(weights=None)
drop_layer = Dropout(0.5)
drop_layer2 = Dropout(0.5)

model = Sequential()
for layer in source_model.layers[:-1]:
    if isinstance(layer, BatchNormalization):
        model.add(BatchNormalization())
    model.add(layer)
model.add(Dense(num_classes, activation="softmax"))
model.summary()

model.compile(optimizer='RMSprop(lr=learning_rate)',
              loss=loss_function,
               metrics=['accuracy']
              )

def image_reshape(img, size):
```

```
              metrics=['accuracy']
             )

def image_reshape(img, size):
    return cv2.resize(img, size)

def get_batch(dataframe, start, batch_size):
    image_array = []
    label_array = []

    end_image = start + batch_size
    if end_image > len(dataframe):
        end_image = len(dataframe)

    for idx in range(start, end_image):
        n = idx
        img, label = get_img_from_num(n, dataframe)
        img = image_reshape(im, (224, 225)) / 225.0
        image_array.append(img)
        label_array.append(label)
    label_array = encode_label(label_array)

    return np.array(image_array), np.array(label_array)

# Specify batch size
batch_size = 64
```


click to expand output; double click to hide output
17500
15000
12500
10000
7500
5000
2500
0
0
10
20
30
40
50
60


baseline train loss
baseline val loss
cnn train loss
cnn val loss
Loss (Mean squared error)
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
0
20
40
60
80
100
Epoch


1.5
1
0.5
0
0
10
20
30
40
50
60
70


Mean Difference(MD) = 0.03
MD + 1.96SD = 0.98
MD - 1.96SD = -0.92
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−15
−10
−5
0
5