Dip Project Code

```
import numpy as np
import os
import cv2
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.models import Model

# Path to dataset
path = "/content/drive/MyDrive/utkface/UTKFace/"
files = os.listdir(path)
size = len(files)
print("Total samples:", size)

# Preprocessing images and labels
images = []
ages = []

for file in files:
    try:
        image = cv2.imread(path + file, 0)  # Grayscale image
        if image is None:  # Handle corrupt files
            print(f"Skipped file: {file} (could not read image)")
            continue
        image = cv2.resize(image, dsize=(64, 64))
        image = image.reshape((image.shape[0], image.shape[1], 1))
        images.append(image)
        split_var = file.split('_')
        ages.append(int(split_var[0]))
    except Exception as e:
        print(f"Error processing file {file}: {e}")
        continue

# Check if images and ages have the same length
assert len(images) == len(ages), "Mismatch in number of images and ages"

# Visualization of age distribution
x_ages = list(set(ages))
y_ages = [ages.count(i) for i in x_ages]
plt.bar(x_ages, y_ages)
plt.title("Age Distribution in Dataset")
plt.xlabel("Age")
plt.ylabel("Count")
plt.show()
```

```
# Helper function for display
def display(img):
    plt.imshow(img[:, :, 0])
    plt.set_cmap('gray')
    plt.show()

# Age Group Classification (Direct Categories)
def age_group(age):
    group = age // 5
    return min(group, 20)  # Cap group index at 20 (corresponding to 100+ age group)

# Preparing features and targets
num_classes = 21  # Age groups: 0-4, 5-9, ..., 100+
target = np.zeros((len(images),), dtype='int32')  # Integer labels
features = np.zeros((len(images), 64, 64, 1), dtype='float32')

for i in range(len(images)):
    target[i] = age_group(ages[i])
    features[i] = images[i]

features = features / 255.0  # Normalize input images
target = to_categorical(target, num_classes=num_classes)  # One-hot encode

# Train-test split
x_train, x_test, y_train, y_test = train_test_split(features, target, test_size=0.2, shuffle=True)
print("Training samples:", x_train.shape[0])
print("Testing samples:", x_test.shape[0])

# Building the model
inputs = Input(shape=(64, 64, 1))
conv1 = Conv2D(32, kernel_size=(3, 3), activation='relu')(inputs)
conv2 = Conv2D(64, kernel_size=(3, 3), activation='relu')(conv1)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv2)
conv3 = Conv2D(128, kernel_size=(3, 3), activation='relu')(pool1)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv3)
x = Dropout(0.25)(pool2)
flat = Flatten()(x)

# Fully connected layers
dense1 = Dense(128, activation='relu')(flat)
dropout1 = Dropout(0.5)(dense1)
dense2 = Dense(64, activation='relu')(dropout1)
dropout2 = Dropout(0.5)(dense2)

# Output layer for classification
outputs = Dense(num_classes, activation='softmax')(dropout2)

# Creating the model
model = Model(inputs=inputs, outputs=outputs)
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Training the model
h = model.fit(
    x_train,
    y_train,
    validation_data=(x_test, y_test),
    epochs=25,
    batch_size=128,
    shuffle=True
)

# Plotting training and validation loss
plt.plot(h.history['loss'], label='Train Loss')
plt.plot(h.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper left')
plt.show()

# Function to map predictions to age ranges
def get_age(prediction):
    group = np.argmax(prediction)  # Get predicted group index
    lower_bound = group * 5
    upper_bound = lower_bound + 4
    if group == 20:
        return "100+"
    return f"{lower_bound}-{upper_bound}"

# Predict and display results
def get_result(sample):
    sample = sample / 255.0  # Normalize input
    prediction = model.predict(np.array([sample]), verbose=0)
    age_range = get_age(prediction)
    print("Predicted Age Range:", age_range)

# Testing the model with some samples
indexes = [0, 10, 50, 100, 200, 300, 400, 500, 600]
for idx in indexes:
    if idx >= len(images):
        print(f"Index {idx} is out of range for the dataset")
        continue
    sample = images[idx]
    display(sample)
    actual_age = ages[idx]
    print("Actual Age:", actual_age)
    get_result(sample)
```