PreprocessParquet

December 11, 2023

```
[ ]: PATH = '../Dataset_GISLR/asl-signs/'
     PROCESSED_OUTPUT_PATH = './Dataset_GISLR_Processed/'
```

```
[ ]: import cv2

     import numpy as np
     import pandas as pd
     from PIL import Image

     from ipywidgets import interact, interactive, fixed, interact_manual
     import ipywidgets as widgets

     import mediapipe as mp
     from mediapipe.framework.formats import landmark_pb2
     mp_drawing = mp.solutions.drawing_utils
     mp_drawing_styles = mp.solutions.drawing_styles
     mp_holistic = mp.solutions.holistic
     import numpy as np
     import matplotlib.pyplot as plt
     import math
     import json
     import os
     import concurrent.futures
```

Iterate over every parquet and do the following: - Find the first 10 frames that contain a consequtive left or right hand - Convert the 10 frames into a mediapipe result structure - Convert each mediapipe result into a numpy array that will be used for training - Store the numpy array into the folder with the matching label/category/sign

```
[ ]: def␣
       ↪draw_landmarks(landmarks,image,show_pose=True,show_face_contour=True,show_face_tesselation=
       ↪
         annotated_image = image.copy()
         results = landmarks
         if show_face_tesselation:
             mp_drawing.draw_landmarks(
                 annotated_image,
                 results.face_landmarks,
```

```python
            mp_holistic.FACEMESH_TESSELATION,
            landmark_drawing_spec=None,
            connection_drawing_spec=mp_drawing_styles
            .get_default_face_mesh_tesselation_style())
    if show_face_contour:
        mp_drawing.draw_landmarks(
            annotated_image,
            results.face_landmarks,
            mp_holistic.FACEMESH_CONTOURS,
            landmark_drawing_spec=None,
            connection_drawing_spec=mp_drawing_styles
            .get_default_face_mesh_contours_style())
    if show_pose:
        mp_drawing.draw_landmarks(
            annotated_image,
            results.pose_landmarks,
            mp_holistic.POSE_CONNECTIONS,
            landmark_drawing_spec=mp_drawing_styles.
            get_default_pose_landmarks_style())
    if show_left_hand:
        mp_drawing.draw_landmarks(
            annotated_image,
            results.left_hand_landmarks,
            mp_holistic.HAND_CONNECTIONS,
            landmark_drawing_spec=mp_drawing_styles
            .get_default_hand_landmarks_style())
    if show_right_hand:
        mp_drawing.draw_landmarks(
            annotated_image,
            results.right_hand_landmarks,
            mp_holistic.HAND_CONNECTIONS,
            landmark_drawing_spec=mp_drawing_styles
            .get_default_hand_landmarks_style())
    return annotated_image

def display_image(img):
    plt.imshow(img)
    plt.axis('off')  # Turn off the axis
    plt.show()
```

[ ]:
```python
def get_avg(example_landmark):
    filtered_landmarks = example_landmark.dropna(subset=["x", "y", "z"])
    filtered_landmarks = filtered_landmarks[(filtered_landmarks[["x", "y",␣
 ↪"z"]] != 0).all(axis=1)]

    # Get the number of landmarks with x, y, z data per type
    landmarks_count = filtered_landmarks["type"].value_counts()
```

```
    meta = landmarks_count.to_dict()
    meta["frames"] = filtered_landmarks["frame"].nunique()

    # Identify unique frames with left and right hand landmarks
    left_hand_frames = filtered_landmarks[filtered_landmarks['type'] ==␣
↪'left_hand']['frame'].nunique()
    right_hand_frames = filtered_landmarks[filtered_landmarks['type'] ==␣
↪'right_hand']['frame'].nunique()
    print(f"Left hand frames: {left_hand_frames}")
    print(f"Right hand frames: {right_hand_frames}")
```

[ ]:
```
from mediapipe.framework.formats import landmark_pb2

class Landmarks(object):
    pass

def get_landmarks_from_parquet(pf,frame):
    f = pf[pf.frame == frame]
    face = landmark_pb2.NormalizedLandmarkList()
    for t in f[f.type=='face'][['x','y','z']].itertuples(index=False):
        face.landmark.add(x=t.x,y=t.y,z=t.z)
    pose = landmark_pb2.NormalizedLandmarkList()
    for t in f[f.type=='pose'][['x','y','z']].itertuples(index=False):
        pose.landmark.add(x=t.x,y=t.y,z=t.z)
    left_hand = landmark_pb2.NormalizedLandmarkList()
    for t in f[f.type=='left_hand'][['x','y','z']].itertuples(index=False):
        left_hand.landmark.add(x=t.x,y=t.y,z=t.z)
    right_hand = landmark_pb2.NormalizedLandmarkList()
    for t in f[f.type=='right_hand'][['x','y','z']].itertuples(index=False):
        right_hand.landmark.add(x=t.x,y=t.y,z=t.z)
    result = Landmarks()
    result.face_landmarks = face
    result.pose_landmarks = pose
    result.left_hand_landmarks = left_hand
    result.right_hand_landmarks = right_hand
    return result
```

[ ]:
```
df = pd.read_csv(PATH + 'train.csv')
df.head()
```

[ ]:
```
                                        path  participant_id  sequence_id  \
0  train_landmark_files/26734/1000035562.parquet           26734   1000035562
1  train_landmark_files/28656/1000106739.parquet           28656   1000106739
2   train_landmark_files/16069/100015657.parquet           16069    100015657
3  train_landmark_files/25571/1000210073.parquet           25571   1000210073
4  train_landmark_files/62590/1000240708.parquet           62590   1000240708
```

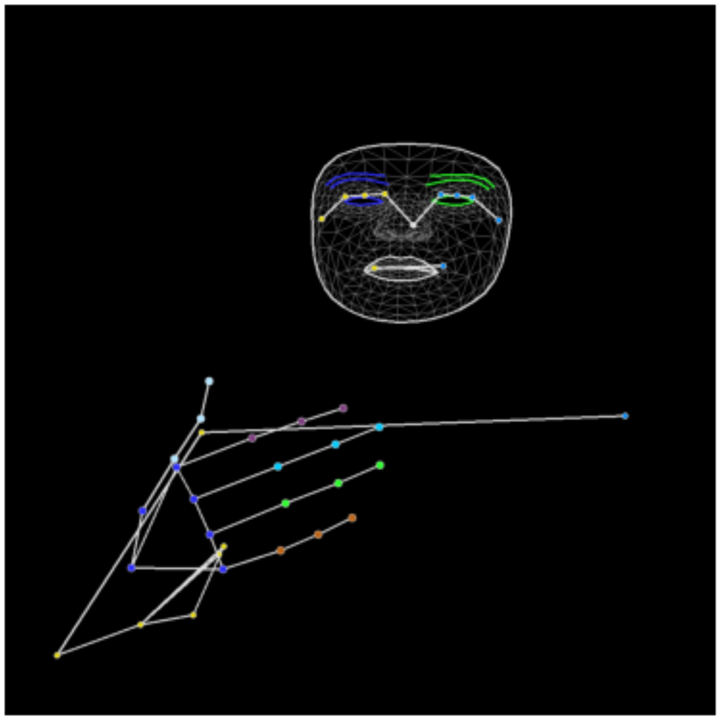
```
    sign
0   blow
1   wait
2  cloud
3   bird
4   owie
```

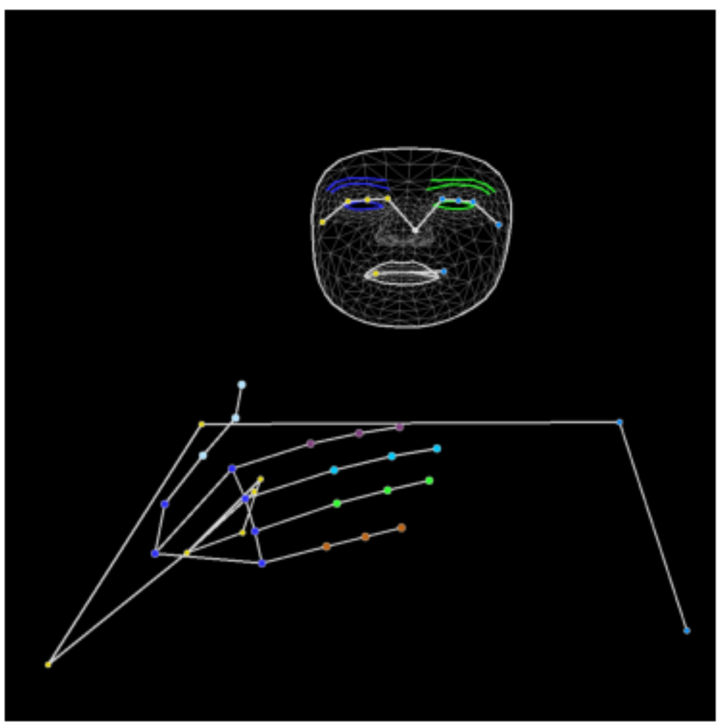
```
[ ]: def decide_which_array_to_use(left_hand_frames, right_hand_frames):
         if left_hand_frames and right_hand_frames:
             # right hand is more important
             return right_hand_frames
         elif left_hand_frames:
             return left_hand_frames
         elif right_hand_frames:
             return right_hand_frames
         else:
             return None

     def are_landmarks_valid(landmarks):
         return all(not math.isnan(landmark.x) and not math.isnan(landmark.y) and␣
      ↪not math.isnan(landmark.z) for landmark in landmarks.landmark)

     def display_all_frames(landmarks_array):
         for i in range(len(landmarks_array)):
             if are_landmarks_valid(landmarks_array[i].pose_landmarks):
                 annotated_image = np.zeros((1024,1024,3),dtype=np.uint8)
                 annotated_image = draw_landmarks(landmarks_array[i],annotated_image)
                 display_image(annotated_image)

     def find_first_10_consecutive_frames(landmarks_array, hand_type):
         consecutive_frames = []
         first_10_frames = []

         for landmarks in landmarks_array:
             if hand_type == "left":
                 has_valid_hand = are_landmarks_valid(landmarks.left_hand_landmarks)
             elif hand_type == "right":
                 has_valid_hand = are_landmarks_valid(landmarks.right_hand_landmarks)
             else:
                 raise ValueError("Hand type must be 'left' or 'right'")

             if has_valid_hand:
                 consecutive_frames.append(landmarks)
                 if len(consecutive_frames) == 10:
                     first_10_frames = consecutive_frames.copy()
                     break
             else:
```

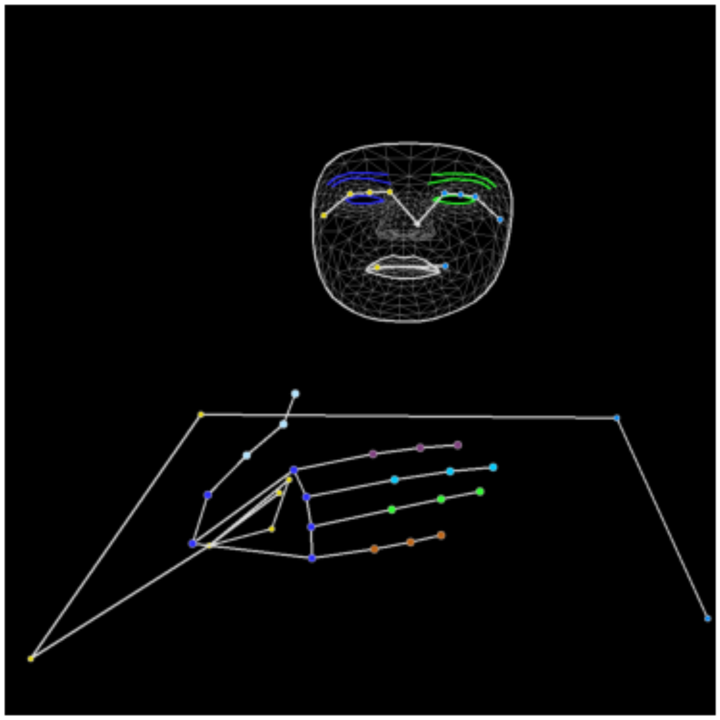
```
            consecutive_frames = []

    return first_10_frames

def process_parquet_file(parquet_file):
    pf = pd.read_parquet(PATH + parquet_file)
    frame_numbers = pf.frame.unique()
    landmarks_array = [get_landmarks_from_parquet(pf, frame) for frame in␣
 ↪frame_numbers]

    first_10_frames_with_left_hand =␣
 ↪find_first_10_consecutive_frames(landmarks_array, "left")
    first_10_frames_with_right_hand =␣
 ↪find_first_10_consecutive_frames(landmarks_array, "right")
    chosen_frames = decide_which_array_to_use(first_10_frames_with_left_hand,␣
 ↪first_10_frames_with_right_hand)
    return chosen_frames
```

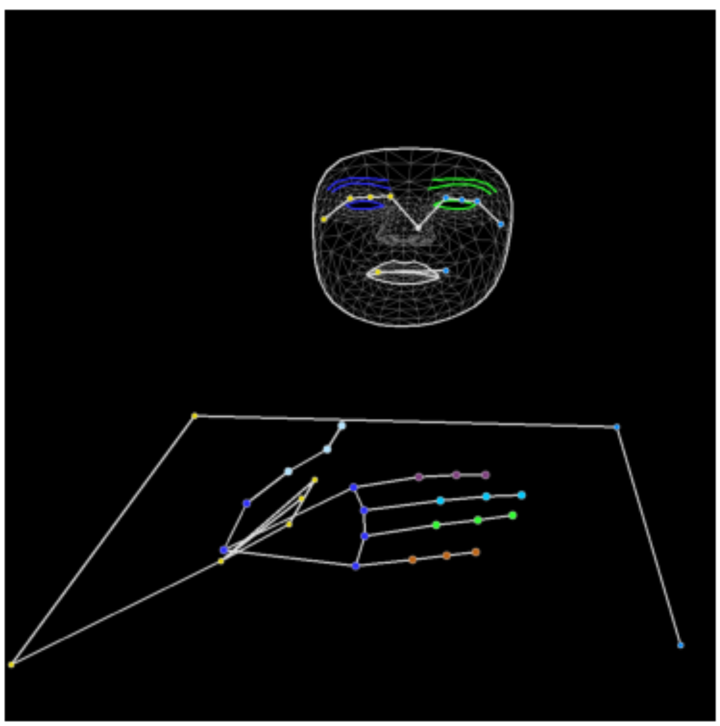
[ ]:
```
def display_index(index):
    parquet_file = df.iloc[index].path
    print(f"Sign: {df.iloc[index].sign}")
    chosen_frames = process_parquet_file(parquet_file)
    if chosen_frames is None:
        return None
    display_all_frames(chosen_frames)


display_index(6)
```

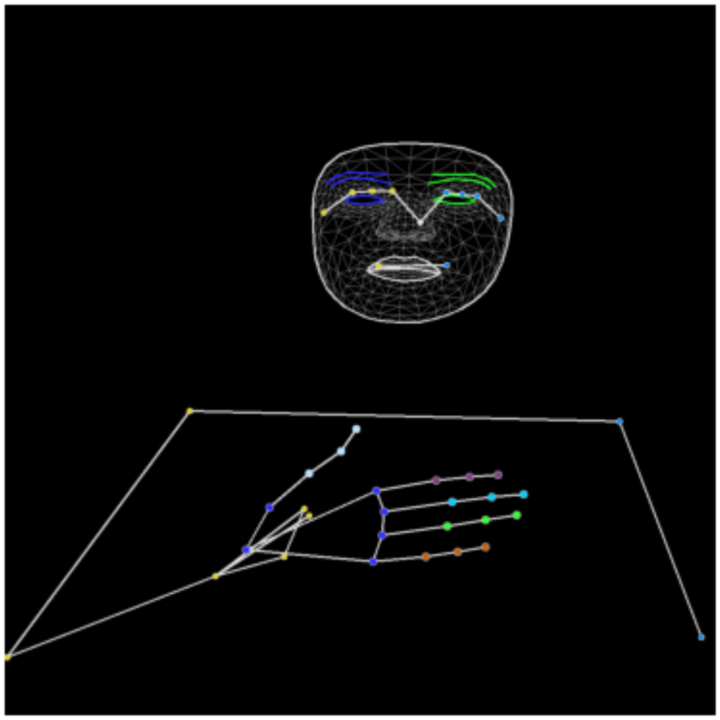
```
Sign: minemy
```

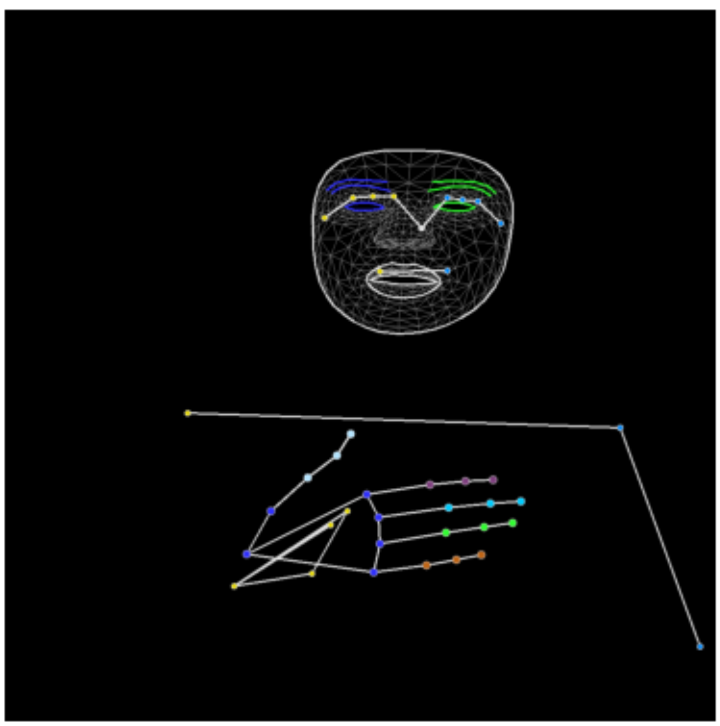


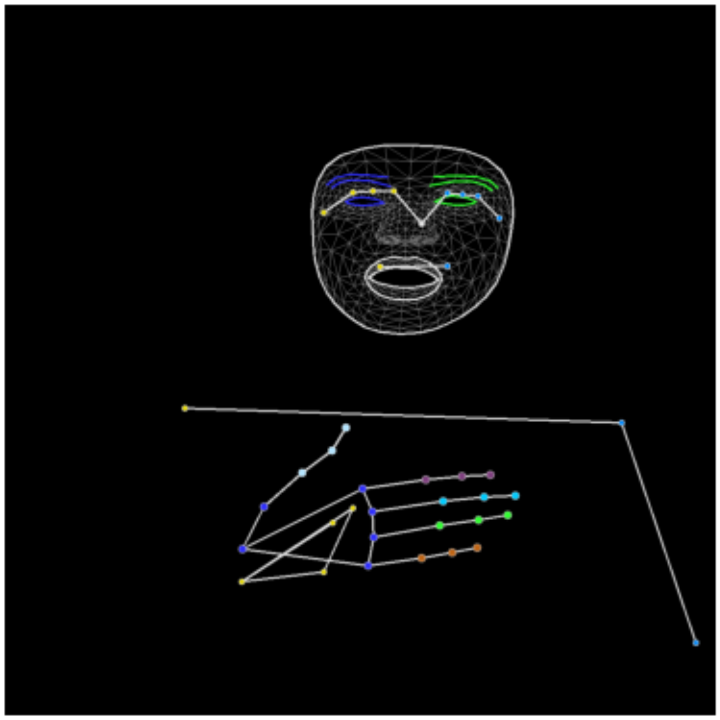


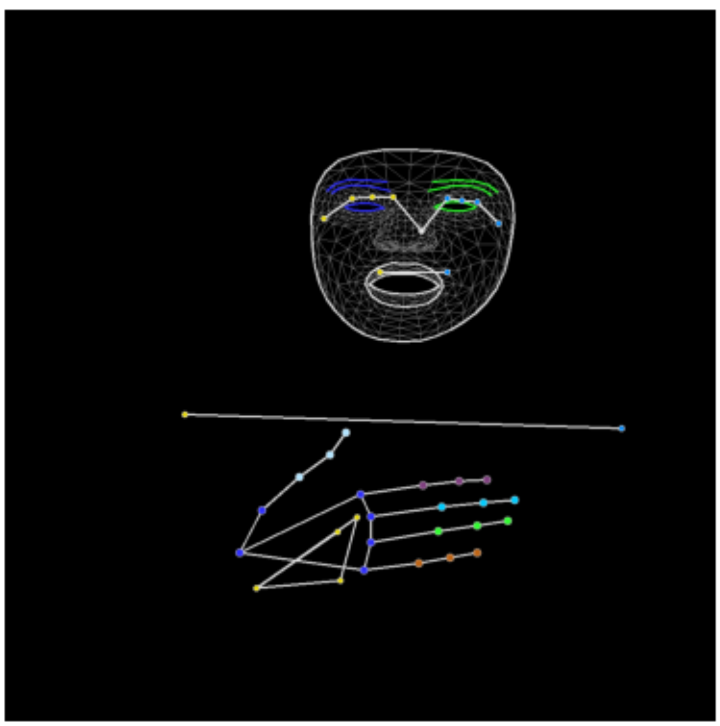


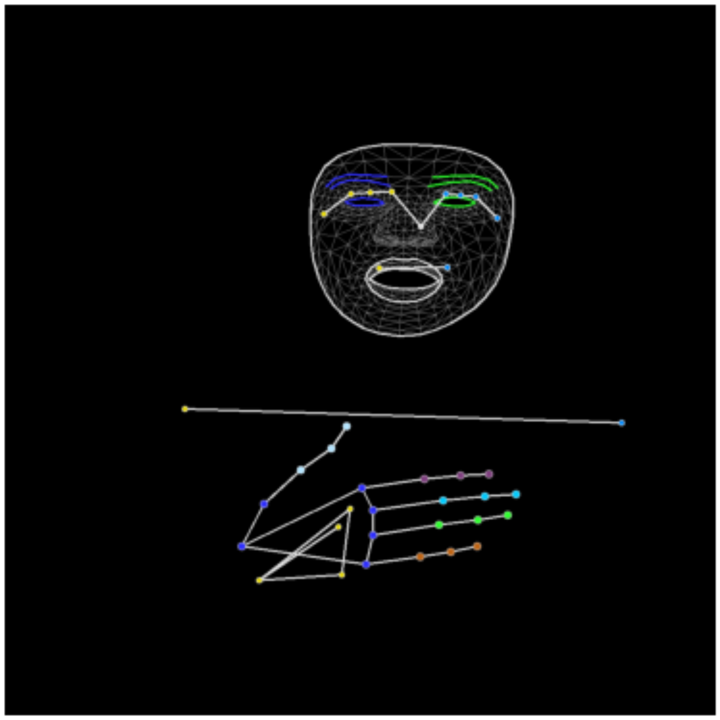


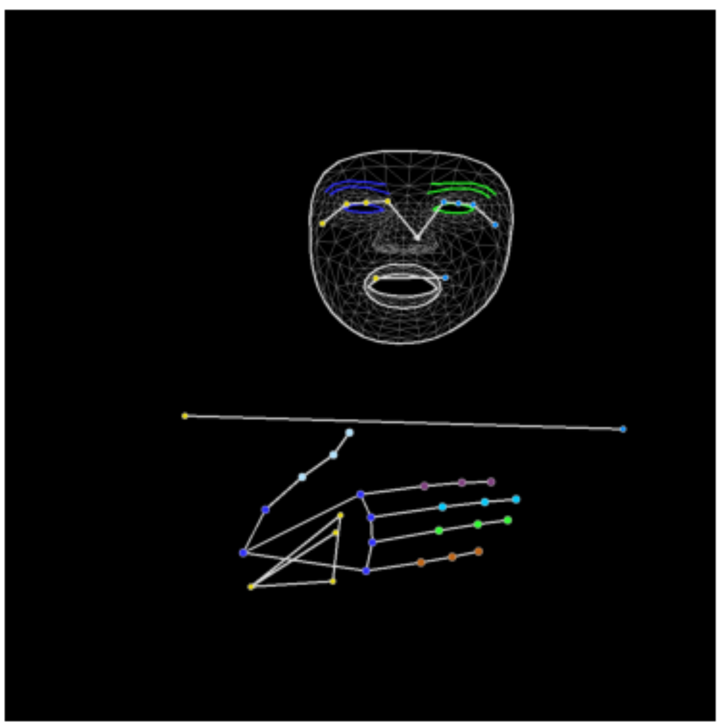












[ ]:
```python
def load_json_file(file_path):
    with open(file_path, 'r') as file:
        data = json.load(file)
    return data
json_file_path = PATH + 'sign_to_prediction_index_map.json'
category_dict = load_json_file(json_file_path)
print(category_dict)
```


```
{'TV': 0, 'after': 1, 'airplane': 2, 'all': 3, 'alligator': 4, 'animal': 5,
'another': 6, 'any': 7, 'apple': 8, 'arm': 9, 'aunt': 10, 'awake': 11,
'backyard': 12, 'bad': 13, 'balloon': 14, 'bath': 15, 'because': 16, 'bed': 17,
'bedroom': 18, 'bee': 19, 'before': 20, 'beside': 21, 'better': 22, 'bird': 23,
'black': 24, 'blow': 25, 'blue': 26, 'boat': 27, 'book': 28, 'boy': 29,
'brother': 30, 'brown': 31, 'bug': 32, 'bye': 33, 'callonphone': 34, 'can': 35,
'car': 36, 'carrot': 37, 'cat': 38, 'cereal': 39, 'chair': 40, 'cheek': 41,
'child': 42, 'chin': 43, 'chocolate': 44, 'clean': 45, 'close': 46, 'closet':
47, 'cloud': 48, 'clown': 49, 'cow': 50, 'cowboy': 51, 'cry': 52, 'cut': 53,
'cute': 54, 'dad': 55, 'dance': 56, 'dirty': 57, 'dog': 58, 'doll': 59,
'donkey': 60, 'down': 61, 'drawer': 62, 'drink': 63, 'drop': 64, 'dry': 65,
'dryer': 66, 'duck': 67, 'ear': 68, 'elephant': 69, 'empty': 70, 'every': 71,
'eye': 72, 'face': 73, 'fall': 74, 'farm': 75, 'fast': 76, 'feet': 77, 'find':
78, 'fine': 79, 'finger': 80, 'finish': 81, 'fireman': 82, 'first': 83, 'fish':
84, 'flag': 85, 'flower': 86, 'food': 87, 'for': 88, 'frenchfries': 89, 'frog':
90, 'garbage': 91, 'gift': 92, 'giraffe': 93, 'girl': 94, 'give': 95,
'glasswindow': 96, 'go': 97, 'goose': 98, 'grandma': 99, 'grandpa': 100,
'grass': 101, 'green': 102, 'gum': 103, 'hair': 104, 'happy': 105, 'hat': 106,
'hate': 107, 'have': 108, 'haveto': 109, 'head': 110, 'hear': 111, 'helicopter':
112, 'hello': 113, 'hen': 114, 'hesheit': 115, 'hide': 116, 'high': 117, 'home':
118, 'horse': 119, 'hot': 120, 'hungry': 121, 'icecream': 122, 'if': 123,
'into': 124, 'jacket': 125, 'jeans': 126, 'jump': 127, 'kiss': 128, 'kitty':
129, 'lamp': 130, 'later': 131, 'like': 132, 'lion': 133, 'lips': 134, 'listen':
135, 'look': 136, 'loud': 137, 'mad': 138, 'make': 139, 'man': 140, 'many': 141,
'milk': 142, 'minemy': 143, 'mitten': 144, 'mom': 145, 'moon': 146, 'morning':
147, 'mouse': 148, 'mouth': 149, 'nap': 150, 'napkin': 151, 'night': 152, 'no':
153, 'noisy': 154, 'nose': 155, 'not': 156, 'now': 157, 'nuts': 158, 'old': 159,
'on': 160, 'open': 161, 'orange': 162, 'outside': 163, 'owie': 164, 'owl': 165,
'pajamas': 166, 'pen': 167, 'pencil': 168, 'penny': 169, 'person': 170, 'pig':
171, 'pizza': 172, 'please': 173, 'police': 174, 'pool': 175, 'potty': 176,
'pretend': 177, 'pretty': 178, 'puppy': 179, 'puzzle': 180, 'quiet': 181,
'radio': 182, 'rain': 183, 'read': 184, 'red': 185, 'refrigerator': 186, 'ride':
187, 'room': 188, 'sad': 189, 'same': 190, 'say': 191, 'scissors': 192, 'see':
193, 'shhh': 194, 'shirt': 195, 'shoe': 196, 'shower': 197, 'sick': 198,
'sleep': 199, 'sleepy': 200, 'smile': 201, 'snack': 202, 'snow': 203, 'stairs':
204, 'stay': 205, 'sticky': 206, 'store': 207, 'story': 208, 'stuck': 209,
'sun': 210, 'table': 211, 'talk': 212, 'taste': 213, 'thankyou': 214, 'that':
215, 'there': 216, 'think': 217, 'thirsty': 218, 'tiger': 219, 'time': 220,
'tomorrow': 221, 'tongue': 222, 'tooth': 223, 'toothbrush': 224, 'touch': 225,
'toy': 226, 'tree': 227, 'uncle': 228, 'underwear': 229, 'up': 230, 'vacuum':
```

```
231, 'wait': 232, 'wake': 233, 'water': 234, 'wet': 235, 'weus': 236, 'where':
237, 'white': 238, 'who': 239, 'why': 240, 'will': 241, 'wolf': 242, 'yellow':
243, 'yes': 244, 'yesterday': 245, 'yourself': 246, 'yucky': 247, 'zebra': 248,
'zipper': 249}
```

[ ]:
```
# size of df
print(len(df))
NUM_TO_TRAIN = len(df)
print(NUM_TO_TRAIN)
```

```
94477
94477
```

[ ]:
```
def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.
 ↪pose_landmarks.landmark]).flatten() if results.pose_landmarks else np.
 ↪zeros(33*4)
    face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.
 ↪landmark]).flatten() if results.face_landmarks else np.zeros(468*3)
    lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.
 ↪landmark]).flatten() if results.left_hand_landmarks else np.zeros(21*3)
    rh = np.array([[res.x, res.y, res.z] for res in results.
 ↪right_hand_landmarks.landmark]).flatten() if results.right_hand_landmarks␣
 ↪else np.zeros(21*3)
    return np.concatenate([pose, face, lh, rh])


def process_and_save_sequences(index, output_folder):
    parquet_file = df.iloc[index].path
    label = df.iloc[index].sign
    name = df.iloc[index].sequence_id

    label_folder = os.path.join(output_folder, label)
    os.makedirs(label_folder, exist_ok=True)

    # make file name with name
    file_name = str(name) + ".npy"
    seq_file = os.path.join(label_folder, file_name)
    # print(f"Processing {seq_file}...")

    # check if file exists
    if os.path.isfile(seq_file):
        # print(f"File {seq_file} already exists. Skipping...")
        return None

    chosen_frames = process_parquet_file(parquet_file)
    if chosen_frames is None:
        return None
```

```
    window = []
    for frame in chosen_frames:
        window.append(extract_keypoints(frame))

    np.save(seq_file, np.array(window))

max_threads = os.cpu_count()  # Specify the number of threads you want to use
print(f"Using {max_threads} threads")
with concurrent.futures.ThreadPoolExecutor(max_workers=max_threads) as executor:
    futures = {executor.submit(process_and_save_sequences, i,␣
 ↪PROCESSED_OUTPUT_PATH): i for i in range(NUM_TO_TRAIN)}
    concurrent.futures.wait(futures)
```

```
Using 16 threads
```

Train

December 11, 2023

[ ]:
```
PATH = '../Dataset_GISLR/asl-signs/'
PROCESSED_OUTPUT_PATH = './Dataset_GISLR_Processed/'
MODEL_VERSION = 'high-perform-test'
save_low_performers = True
save_high_performers = False
```

[ ]:
```
import cv2

import numpy as np
import pandas as pd
from PIL import Image

from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets

import mediapipe as mp
from mediapipe.framework.formats import landmark_pb2
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_holistic = mp.solutions.holistic
import numpy as np
import matplotlib.pyplot as plt
import math
import json
import os
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import TensorBoard
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.mixed_precision import set_global_policy
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import classification_report
import numpy as np
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, mean_squared_error, mean_absolute_error
```

```
2023-12-02 08:37:44.380617: E
tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:9342] Unable to
register cuDNN factory: Attempting to register factory for plugin cuDNN when one
has already been registered
2023-12-02 08:37:44.380665: E
tensorflow/compiler/xla/stream_executor/cuda/cuda_fft.cc:609] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2023-12-02 08:37:44.380683: E
tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:1518] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2023-12-02 08:37:44.385081: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy
version >=1.17.3 and <1.25.0 is required for this version of SciPy (detected
version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Load the Labels from the PreProcessed Data. Optionally filter out labels to remove any identified low performing labels.

[ ]:
```
signs_to_train = [name for name in os.listdir(PROCESSED_OUTPUT_PATH) if os.path.
↪isdir(os.path.join(PROCESSED_OUTPUT_PATH, name))]

low_performing_labels_file_path = 'low_performing_labels.npy'
low_performing_labels = np.array([])
if os.path.exists(low_performing_labels_file_path):
    low_performing_labels = np.load(low_performing_labels_file_path,␣
↪allow_pickle=True)
print(len(low_performing_labels))

high_performing_labels_file_path = 'high_performing_labels.npy'
high_performing_labels = np.array([])
if os.path.exists(high_performing_labels_file_path):
    high_performing_labels = np.load(high_performing_labels_file_path,␣
↪allow_pickle=True)
print(len(high_performing_labels))

# remove low performing labels from signs_to_train
signs_to_train = [x for x in signs_to_train if x not in low_performing_labels]
print(len(signs_to_train))
```

```
# remove all non high performers
if (len(high_performing_labels) > 0):
    signs_to_train = [x for x in signs_to_train if x in high_performing_labels]
    print(len(signs_to_train))

signs_to_train.sort()
actions = np.array(signs_to_train)
print(actions)
```

```
197
53
['aunt' 'bird' 'black' 'brother' 'brown' 'bug' 'callonphone' 'cheek'
 'clown' 'cow' 'cute' 'dad' 'doll' 'donkey' 'drink' 'ear' 'eye' 'feet'
 'find' 'fireman' 'flower' 'for' 'frog' 'grandpa' 'grass' 'gum' 'hair'
 'hen' 'home' 'horse' 'lamp' 'mad' 'mom' 'mouse' 'nose' 'owl' 'pig'
 'police' 'radio' 'see' 'shhh' 'shirt' 'sick' 'stairs' 'stuck' 'taste'
 'thirsty' 'tiger' 'uncle' 'water' 'who' 'yucky' 'zebra']
```

Load all of the pre processed data

[ ]:
```
label_map = {label:num for num, label in enumerate(actions)}

sequences, labels = [], []

for label_folder in os.listdir(PROCESSED_OUTPUT_PATH):
    label = label_folder  # Use the folder name as the label
    if (label not in signs_to_train):
        continue

    if (label in low_performing_labels):
        print("Skipping label: ", label)
        continue

    label_folder_path = os.path.join(PROCESSED_OUTPUT_PATH, label_folder)

    # Iterate through the files in the label folder (each file is a sequence)
    for sequence_file in os.listdir(label_folder_path):
        if sequence_file.endswith('.npy'):
            # Load the sequence from the file
            sequence = np.load(os.path.join(label_folder_path, sequence_file))

            # Append the sequence and label to the respective lists
            sequences.append(sequence)
            labels.append(label_map[label])
```

[ ]:
```
X = np.array(sequences)
print(X.shape)
```

```
(13288, 10, 1662)
```

[ ]:
```python
y = to_categorical(labels).astype(int)
print(y.shape)
```

```
(13288, 53)
```

Test train split all of the data

[ ]:
```python
# First, split the data into training and a combined validation/test set
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3)  #␣
 ↪Adjust the test_size as needed

# Now, split the combined validation/test set into separate validation and test␣
 ↪sets
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5) ␣
 ↪# This will split the remaining 30% into two 15% sets
```

[ ]:
```python
print(y_test.shape)
print(y_val.shape)
print(y_train.shape)
```

```
(1994, 53)
(1993, 53)
(9301, 53)
```

Replace all nan else the model will not train correctly

[ ]:
```python
# Check for NaN values in the datasets
print(f"NaNs in X_train: {np.isnan(X_train).any()}, NaNs in y_train: {np.
 ↪isnan(y_train).any()}")
print(f"NaNs in X_val: {np.isnan(X_val).any()}, NaNs in y_val: {np.isnan(y_val).
 ↪any()}")
print(f"NaNs in X_test: {np.isnan(X_test).any()}, NaNs in y_test: {np.
 ↪isnan(y_test).any()}")

# Replace NaN values in the datasets
X_train = np.nan_to_num(X_train)
y_train = np.nan_to_num(y_train)
X_val = np.nan_to_num(X_val)
y_val = np.nan_to_num(y_val)
X_test = np.nan_to_num(X_test)
y_test = np.nan_to_num(y_test)
```

```
NaNs in X_train: True, NaNs in y_train: False
NaNs in X_val: True, NaNs in y_val: False
NaNs in X_test: True, NaNs in y_test: False
```

Build the model WIP

[ ]:

```
# v1
# model = Sequential()
# model.add(LSTM(64, return_sequences=True, activation='relu',␣
↪input_shape=(10,1662)))
# model.add(LSTM(128, return_sequences=True, activation='relu'))
# model.add(LSTM(64, return_sequences=False, activation='relu'))
# model.add(Dense(64, activation='relu'))
# model.add(Dense(32, activation='relu'))
# model.add(Dense(actions.shape[0], activation='softmax'))
# model.compile(optimizer='Adam', loss='categorical_crossentropy',␣
↪metrics=['categorical_accuracy'])

frame_length = 1662
num_classes = actions.shape[0]

# v2
model = Sequential()
model.add(LSTM(128, return_sequences=True, activation='relu', input_shape=(10,␣
↪frame_length)))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(LSTM(256, return_sequences=False, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

# Optimizer with Gradient Clipping
# optimizer = Adam(learning_rate=0.0001, clipvalue=0.5)  # Adjust learning rate␣
↪and clipvalue as needed
optimizer = Adam(learning_rate=0.001)

# Compile the model
model.compile(optimizer=optimizer, loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=100, verbose=1,␣
↪mode='min')

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=25,␣
↪verbose=1, mode='min', min_lr=0.00001)
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet
the criteria. It will use a generic GPU kernel as fallback when running on GPU.

2023-12-02 08:37:54.430084: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:880] could not
```

```
open file to read NUMA node: /sys/bus/pci/devices/0000:01:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-12-02 08:37:54.432988: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:880] could not
open file to read NUMA node: /sys/bus/pci/devices/0000:01:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-12-02 08:37:54.433036: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:880] could not
open file to read NUMA node: /sys/bus/pci/devices/0000:01:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-12-02 08:37:54.434702: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:880] could not
open file to read NUMA node: /sys/bus/pci/devices/0000:01:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-12-02 08:37:54.434736: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:880] could not
open file to read NUMA node: /sys/bus/pci/devices/0000:01:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-12-02 08:37:54.434757: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:880] could not
open file to read NUMA node: /sys/bus/pci/devices/0000:01:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-12-02 08:37:54.719876: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:880] could not
open file to read NUMA node: /sys/bus/pci/devices/0000:01:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-12-02 08:37:54.719921: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:880] could not
open file to read NUMA node: /sys/bus/pci/devices/0000:01:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-12-02 08:37:54.719929: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1977] Could not identify NUMA
node of platform GPU id 0, defaulting to 0.  Your kernel may not have been built
with NUMA support.
2023-12-02 08:37:54.719956: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:880] could not
open file to read NUMA node: /sys/bus/pci/devices/0000:01:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-12-02 08:37:54.719971: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1886] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 21472 MB memory:  -> device:
0, name: NVIDIA GeForce RTX 4090, pci bus id: 0000:01:00.0, compute capability:
8.9
2023-12-02 08:37:55.285017: I tensorflow/tsl/platform/default/subprocess.cc:304]
Start cannot spawn child process: No such file or directory

WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it doesn't meet
the criteria. It will use a generic GPU kernel as fallback when running on GPU.
```

Train the model

[ ]:
```
# Enable mixed precision
set_global_policy('mixed_float16')

epochs = 1000  # Adjust number of epochs as needed
batch_size = 1024

history = model.fit(
    X_train, y_train,
    epochs=epochs,
    batch_size=batch_size,
    validation_data=(X_val, y_val),
    callbacks=[early_stopping, reduce_lr]
)
```

```
INFO:tensorflow:Mixed precision compatibility check (mixed_float16): OK
Your GPU will likely run quickly with dtype policy mixed_float16 as it has
compute capability of at least 7.0. Your GPU: NVIDIA GeForce RTX 4090, compute
capability 8.9
```

```
2023-12-02 08:37:55.529626: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:880] could not
open file to read NUMA node: /sys/bus/pci/devices/0000:01:00.0/numa_node
Your kernel may have been built without NUMA support.
```

```
Epoch 1/1000
```

```
2023-12-02 08:38:00.429825: I tensorflow/compiler/xla/service/service.cc:168]
XLA service 0x7fb908296f30 initialized for platform CUDA (this does not
guarantee that XLA will be used). Devices:
2023-12-02 08:38:00.429869: I tensorflow/compiler/xla/service/service.cc:176]
StreamExecutor device (0): NVIDIA GeForce RTX 4090, Compute Capability 8.9
2023-12-02 08:38:00.434596: I
tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR
crash reproducer, set env var `MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.
2023-12-02 08:38:00.449839: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:442] Loaded cuDNN
version 8700
2023-12-02 08:38:00.518010: I ./tensorflow/compiler/jit/device_compiler.h:186]
Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
```

```
10/10 [==============================] - 7s 167ms/step - loss: 3.9674 -
accuracy: 0.0229 - val_loss: 3.9901 - val_accuracy: 0.0311 - lr: 0.0010
Epoch 2/1000
10/10 [==============================] - 1s 90ms/step - loss: 3.8960 - accuracy:
0.0342 - val_loss: 3.9989 - val_accuracy: 0.0216 - lr: 0.0010
Epoch 3/1000
10/10 [==============================] - 1s 78ms/step - loss: 3.7596 - accuracy:
```

```
0.0483 - val_loss: 4.0058 - val_accuracy: 0.0211 - lr: 0.0010
Epoch 4/1000
10/10 [==============================] - 1s 89ms/step - loss: 3.6526 - accuracy:
0.0654 - val_loss: 3.9937 - val_accuracy: 0.0321 - lr: 0.0010
Epoch 5/1000
10/10 [==============================] - 1s 81ms/step - loss: 3.5115 - accuracy:
0.0840 - val_loss: 3.7521 - val_accuracy: 0.0396 - lr: 0.0010
Epoch 6/1000
10/10 [==============================] - 1s 76ms/step - loss: 3.3786 - accuracy:
0.1028 - val_loss: 3.7308 - val_accuracy: 0.0447 - lr: 0.0010
Epoch 7/1000
10/10 [==============================] - 1s 84ms/step - loss: 3.2247 - accuracy:
0.1229 - val_loss: 3.6293 - val_accuracy: 0.0542 - lr: 0.0010
Epoch 8/1000
10/10 [==============================] - 1s 72ms/step - loss: 3.0981 - accuracy:
0.1404 - val_loss: 3.5763 - val_accuracy: 0.0833 - lr: 0.0010
Epoch 9/1000
10/10 [==============================] - 1s 77ms/step - loss: 3.0148 - accuracy:
0.1584 - val_loss: 3.5610 - val_accuracy: 0.0758 - lr: 0.0010
Epoch 10/1000
10/10 [==============================] - 1s 75ms/step - loss: 2.8824 - accuracy:
0.1866 - val_loss: 3.4611 - val_accuracy: 0.0682 - lr: 0.0010
Epoch 11/1000
10/10 [==============================] - 1s 73ms/step - loss: 2.7910 - accuracy:
0.2046 - val_loss: 3.6399 - val_accuracy: 0.0702 - lr: 0.0010
Epoch 12/1000
10/10 [==============================] - 1s 75ms/step - loss: 2.7474 - accuracy:
0.2120 - val_loss: 3.4201 - val_accuracy: 0.0838 - lr: 0.0010
Epoch 13/1000
10/10 [==============================] - 1s 75ms/step - loss: 2.5985 - accuracy:
0.2437 - val_loss: 4.9064 - val_accuracy: 0.0592 - lr: 0.0010
Epoch 14/1000
10/10 [==============================] - 1s 74ms/step - loss: 2.5362 - accuracy:
0.2666 - val_loss: 3.5530 - val_accuracy: 0.0828 - lr: 0.0010
Epoch 15/1000
10/10 [==============================] - 1s 78ms/step - loss: 2.4691 - accuracy:
0.2720 - val_loss: 3.1892 - val_accuracy: 0.1445 - lr: 0.0010
Epoch 16/1000
10/10 [==============================] - 1s 83ms/step - loss: 2.4212 - accuracy:
0.2949 - val_loss: 3.5145 - val_accuracy: 0.0948 - lr: 0.0010
Epoch 17/1000
10/10 [==============================] - 1s 77ms/step - loss: 2.3630 - accuracy:
0.3104 - val_loss: 4.0417 - val_accuracy: 0.0462 - lr: 0.0010
Epoch 18/1000
10/10 [==============================] - 1s 75ms/step - loss: 2.2812 - accuracy:
0.3257 - val_loss: 3.4919 - val_accuracy: 0.0973 - lr: 0.0010
Epoch 19/1000
10/10 [==============================] - 1s 74ms/step - loss: 2.2174 - accuracy:
```

```
0.3410 - val_loss: 3.4911 - val_accuracy: 0.0768 - lr: 0.0010
Epoch 20/1000
10/10 [==============================] - 1s 76ms/step - loss: 2.1749 - accuracy:
0.3528 - val_loss: 4.7066 - val_accuracy: 0.0622 - lr: 0.0010
Epoch 21/1000
10/10 [==============================] - 1s 82ms/step - loss: 2.1075 - accuracy:
0.3792 - val_loss: 3.6192 - val_accuracy: 0.0973 - lr: 0.0010
Epoch 22/1000
10/10 [==============================] - 1s 78ms/step - loss: 2.0575 - accuracy:
0.3858 - val_loss: 4.0382 - val_accuracy: 0.0853 - lr: 0.0010
Epoch 23/1000
10/10 [==============================] - 1s 92ms/step - loss: 2.0182 - accuracy:
0.3997 - val_loss: 3.4984 - val_accuracy: 0.1209 - lr: 0.0010
Epoch 24/1000
10/10 [==============================] - 1s 79ms/step - loss: 2.0045 - accuracy:
0.4036 - val_loss: 3.6899 - val_accuracy: 0.0993 - lr: 0.0010
Epoch 25/1000
10/10 [==============================] - 1s 77ms/step - loss: 1.9358 - accuracy:
0.4255 - val_loss: 4.0941 - val_accuracy: 0.0768 - lr: 0.0010
Epoch 26/1000
10/10 [==============================] - 1s 72ms/step - loss: 1.9074 - accuracy:
0.4374 - val_loss: 6.8642 - val_accuracy: 0.0336 - lr: 0.0010
Epoch 27/1000
10/10 [==============================] - 1s 73ms/step - loss: 1.9023 - accuracy:
0.4295 - val_loss: 4.4013 - val_accuracy: 0.0587 - lr: 0.0010
Epoch 28/1000
10/10 [==============================] - 1s 74ms/step - loss: 1.8373 - accuracy:
0.4539 - val_loss: 3.5164 - val_accuracy: 0.0848 - lr: 0.0010
Epoch 29/1000
10/10 [==============================] - 1s 72ms/step - loss: 1.8272 - accuracy:
0.4513 - val_loss: 5.1641 - val_accuracy: 0.0467 - lr: 0.0010
Epoch 30/1000
10/10 [==============================] - 1s 83ms/step - loss: 1.8552 - accuracy:
0.4467 - val_loss: 3.1058 - val_accuracy: 0.1681 - lr: 0.0010
Epoch 31/1000
10/10 [==============================] - 1s 77ms/step - loss: 1.7859 - accuracy:
0.4711 - val_loss: 3.5024 - val_accuracy: 0.1706 - lr: 0.0010
Epoch 32/1000
10/10 [==============================] - 1s 74ms/step - loss: 1.7010 - accuracy:
0.4883 - val_loss: 12.4740 - val_accuracy: 0.0426 - lr: 0.0010
Epoch 33/1000
10/10 [==============================] - 1s 71ms/step - loss: 1.7424 - accuracy:
0.4862 - val_loss: 6.0236 - val_accuracy: 0.0748 - lr: 0.0010
Epoch 34/1000
10/10 [==============================] - 1s 76ms/step - loss: 1.6795 - accuracy:
0.4940 - val_loss: 4.4060 - val_accuracy: 0.1099 - lr: 0.0010
Epoch 35/1000
10/10 [==============================] - 1s 73ms/step - loss: 1.6527 - accuracy:
```

```
0.5054 - val_loss: 4.4375 - val_accuracy: 0.0898 - lr: 0.0010
Epoch 36/1000
10/10 [==============================] - 1s 80ms/step - loss: 1.6219 - accuracy:
0.5122 - val_loss: 3.5053 - val_accuracy: 0.1676 - lr: 0.0010
Epoch 37/1000
10/10 [==============================] - 1s 86ms/step - loss: 1.5997 - accuracy:
0.5228 - val_loss: 4.3137 - val_accuracy: 0.1199 - lr: 0.0010
Epoch 38/1000
10/10 [==============================] - 1s 88ms/step - loss: 1.5198 - accuracy:
0.5409 - val_loss: 3.9728 - val_accuracy: 0.1385 - lr: 0.0010
Epoch 39/1000
10/10 [==============================] - 1s 76ms/step - loss: 1.5888 - accuracy:
0.5285 - val_loss: 2.1798 - val_accuracy: 0.3683 - lr: 0.0010
Epoch 40/1000
10/10 [==============================] - 1s 73ms/step - loss: 1.5421 - accuracy:
0.5376 - val_loss: 2.8181 - val_accuracy: 0.2353 - lr: 0.0010
Epoch 41/1000
10/10 [==============================] - 1s 76ms/step - loss: 1.4935 - accuracy:
0.5511 - val_loss: 4.3897 - val_accuracy: 0.1867 - lr: 0.0010
Epoch 42/1000
10/10 [==============================] - 1s 85ms/step - loss: 1.4569 - accuracy:
0.5661 - val_loss: 4.1782 - val_accuracy: 0.1194 - lr: 0.0010
Epoch 43/1000
10/10 [==============================] - 1s 71ms/step - loss: 1.4148 - accuracy:
0.5706 - val_loss: 4.1497 - val_accuracy: 0.1530 - lr: 0.0010
Epoch 44/1000
10/10 [==============================] - 1s 79ms/step - loss: 1.3795 - accuracy:
0.5863 - val_loss: 2.0730 - val_accuracy: 0.4155 - lr: 0.0010
Epoch 45/1000
10/10 [==============================] - 1s 79ms/step - loss: 1.3581 - accuracy:
0.5947 - val_loss: 3.1523 - val_accuracy: 0.2358 - lr: 0.0010
Epoch 46/1000
10/10 [==============================] - 1s 84ms/step - loss: 1.3892 - accuracy:
0.5823 - val_loss: 4.4175 - val_accuracy: 0.1661 - lr: 0.0010
Epoch 47/1000
10/10 [==============================] - 1s 80ms/step - loss: 1.4069 - accuracy:
0.5719 - val_loss: 2.6768 - val_accuracy: 0.3111 - lr: 0.0010
Epoch 48/1000
10/10 [==============================] - 1s 79ms/step - loss: 1.3780 - accuracy:
0.5797 - val_loss: 2.1476 - val_accuracy: 0.3859 - lr: 0.0010
Epoch 49/1000
10/10 [==============================] - 1s 72ms/step - loss: 1.2942 - accuracy:
0.6074 - val_loss: 2.2993 - val_accuracy: 0.3457 - lr: 0.0010
Epoch 50/1000
10/10 [==============================] - 1s 76ms/step - loss: 1.3041 - accuracy:
0.6066 - val_loss: 2.5976 - val_accuracy: 0.3773 - lr: 0.0010
Epoch 51/1000
10/10 [==============================] - 1s 79ms/step - loss: 1.2611 - accuracy:
```

```
0.6214 - val_loss: 2.8531 - val_accuracy: 0.3281 - lr: 0.0010
Epoch 52/1000
10/10 [==============================] - 1s 74ms/step - loss: 1.2445 - accuracy:
0.6214 - val_loss: 2.5329 - val_accuracy: 0.3994 - lr: 0.0010
Epoch 53/1000
10/10 [==============================] - 1s 76ms/step - loss: 1.2473 - accuracy:
0.6227 - val_loss: 4.4819 - val_accuracy: 0.1977 - lr: 0.0010
Epoch 54/1000
10/10 [==============================] - 1s 84ms/step - loss: 1.2691 - accuracy:
0.6158 - val_loss: 3.1625 - val_accuracy: 0.2704 - lr: 0.0010
Epoch 55/1000
10/10 [==============================] - 1s 74ms/step - loss: 1.2232 - accuracy:
0.6283 - val_loss: 4.4479 - val_accuracy: 0.1656 - lr: 0.0010
Epoch 56/1000
10/10 [==============================] - 1s 74ms/step - loss: 1.2235 - accuracy:
0.6313 - val_loss: 3.3525 - val_accuracy: 0.2263 - lr: 0.0010
Epoch 57/1000
10/10 [==============================] - 1s 71ms/step - loss: 1.2259 - accuracy:
0.6277 - val_loss: 2.3473 - val_accuracy: 0.3552 - lr: 0.0010
Epoch 58/1000
10/10 [==============================] - 1s 73ms/step - loss: 1.1917 - accuracy:
0.6386 - val_loss: 4.1074 - val_accuracy: 0.1621 - lr: 0.0010
Epoch 59/1000
10/10 [==============================] - 1s 72ms/step - loss: 1.2050 - accuracy:
0.6334 - val_loss: 2.6640 - val_accuracy: 0.3382 - lr: 0.0010
Epoch 60/1000
10/10 [==============================] - 1s 72ms/step - loss: 1.1889 - accuracy:
0.6423 - val_loss: 2.1666 - val_accuracy: 0.3828 - lr: 0.0010
Epoch 61/1000
10/10 [==============================] - 1s 76ms/step - loss: 1.1912 - accuracy:
0.6421 - val_loss: 2.5516 - val_accuracy: 0.3387 - lr: 0.0010
Epoch 62/1000
10/10 [==============================] - 1s 83ms/step - loss: 1.1493 - accuracy:
0.6549 - val_loss: 2.8114 - val_accuracy: 0.3056 - lr: 0.0010
Epoch 63/1000
10/10 [==============================] - 1s 73ms/step - loss: 1.1174 - accuracy:
0.6605 - val_loss: 2.3602 - val_accuracy: 0.3723 - lr: 0.0010
Epoch 64/1000
10/10 [==============================] - 1s 71ms/step - loss: 1.1263 - accuracy:
0.6586 - val_loss: 2.3414 - val_accuracy: 0.3904 - lr: 0.0010
Epoch 65/1000
10/10 [==============================] - 1s 72ms/step - loss: 1.0839 - accuracy:
0.6768 - val_loss: 1.9748 - val_accuracy: 0.4596 - lr: 0.0010
Epoch 66/1000
10/10 [==============================] - 1s 73ms/step - loss: 1.0652 - accuracy:
0.6753 - val_loss: 3.4216 - val_accuracy: 0.2679 - lr: 0.0010
Epoch 67/1000
10/10 [==============================] - 1s 73ms/step - loss: 1.0615 - accuracy:
```

```
0.6795 - val_loss: 6.3611 - val_accuracy: 0.1445 - lr: 0.0010
Epoch 68/1000
10/10 [==============================] - 1s 74ms/step - loss: 1.0327 - accuracy:
0.6914 - val_loss: 2.8259 - val_accuracy: 0.3583 - lr: 0.0010
Epoch 69/1000
10/10 [==============================] - 1s 72ms/step - loss: 1.0158 - accuracy:
0.6953 - val_loss: 4.2136 - val_accuracy: 0.2303 - lr: 0.0010
Epoch 70/1000
10/10 [==============================] - 1s 75ms/step - loss: 1.0167 - accuracy:
0.6940 - val_loss: 2.2607 - val_accuracy: 0.4596 - lr: 0.0010
Epoch 71/1000
10/10 [==============================] - 1s 82ms/step - loss: 1.0259 - accuracy:
0.6864 - val_loss: 3.1059 - val_accuracy: 0.2840 - lr: 0.0010
Epoch 72/1000
10/10 [==============================] - 1s 75ms/step - loss: 1.0504 - accuracy:
0.6798 - val_loss: 6.2830 - val_accuracy: 0.1189 - lr: 0.0010
Epoch 73/1000
10/10 [==============================] - 1s 74ms/step - loss: 1.0328 - accuracy:
0.6884 - val_loss: 2.7543 - val_accuracy: 0.4104 - lr: 0.0010
Epoch 74/1000
10/10 [==============================] - 1s 71ms/step - loss: 1.0141 - accuracy:
0.6901 - val_loss: 3.5821 - val_accuracy: 0.2935 - lr: 0.0010
Epoch 75/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.9860 - accuracy:
0.7013 - val_loss: 4.3199 - val_accuracy: 0.2283 - lr: 0.0010
Epoch 76/1000
10/10 [==============================] - 1s 73ms/step - loss: 1.0119 - accuracy:
0.6930 - val_loss: 2.5538 - val_accuracy: 0.4134 - lr: 0.0010
Epoch 77/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.9395 - accuracy:
0.7139 - val_loss: 2.2958 - val_accuracy: 0.4541 - lr: 0.0010
Epoch 78/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.9641 - accuracy:
0.7135 - val_loss: 3.3550 - val_accuracy: 0.3557 - lr: 0.0010
Epoch 79/1000
10/10 [==============================] - 1s 76ms/step - loss: 0.9497 - accuracy:
0.7111 - val_loss: 2.4026 - val_accuracy: 0.4134 - lr: 0.0010
Epoch 80/1000
10/10 [==============================] - 1s 87ms/step - loss: 1.0374 - accuracy:
0.6870 - val_loss: 3.5296 - val_accuracy: 0.2393 - lr: 0.0010
Epoch 81/1000
10/10 [==============================] - 1s 81ms/step - loss: 1.0288 - accuracy:
0.6873 - val_loss: 3.0178 - val_accuracy: 0.3377 - lr: 0.0010
Epoch 82/1000
10/10 [==============================] - 1s 76ms/step - loss: 0.9726 - accuracy:
0.7048 - val_loss: 2.3653 - val_accuracy: 0.3954 - lr: 0.0010
Epoch 83/1000
10/10 [==============================] - 1s 71ms/step - loss: 0.9354 - accuracy:
```

```
0.7166 - val_loss: 2.9368 - val_accuracy: 0.3402 - lr: 0.0010
Epoch 84/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.9233 - accuracy:
0.7195 - val_loss: 4.0249 - val_accuracy: 0.2659 - lr: 0.0010
Epoch 85/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.9306 - accuracy:
0.7159 - val_loss: 5.8567 - val_accuracy: 0.1641 - lr: 0.0010
Epoch 86/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.9423 - accuracy:
0.7099 - val_loss: 3.9650 - val_accuracy: 0.3136 - lr: 0.0010
Epoch 87/1000
10/10 [==============================] - 1s 83ms/step - loss: 0.9080 - accuracy:
0.7220 - val_loss: 3.6757 - val_accuracy: 0.4205 - lr: 0.0010
Epoch 88/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.9242 - accuracy:
0.7185 - val_loss: 2.6092 - val_accuracy: 0.3964 - lr: 0.0010
Epoch 89/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.8571 - accuracy:
0.7409 - val_loss: 3.1556 - val_accuracy: 0.3823 - lr: 0.0010
Epoch 90/1000
10/10 [==============================] - ETA: 0s - loss: 0.8515 - accuracy:
0.7431
Epoch 90: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
10/10 [==============================] - 1s 71ms/step - loss: 0.8515 - accuracy:
0.7431 - val_loss: 2.8203 - val_accuracy: 0.3813 - lr: 0.0010
Epoch 91/1000
10/10 [==============================] - 1s 71ms/step - loss: 0.8870 - accuracy:
0.7291 - val_loss: 1.7801 - val_accuracy: 0.5178 - lr: 5.0000e-04
Epoch 92/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.8258 - accuracy:
0.7485 - val_loss: 2.0921 - val_accuracy: 0.5043 - lr: 5.0000e-04
Epoch 93/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.7697 - accuracy:
0.7660 - val_loss: 1.5526 - val_accuracy: 0.5951 - lr: 5.0000e-04
Epoch 94/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.7515 - accuracy:
0.7691 - val_loss: 2.0692 - val_accuracy: 0.4972 - lr: 5.0000e-04
Epoch 95/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.7604 - accuracy:
0.7707 - val_loss: 1.8038 - val_accuracy: 0.5640 - lr: 5.0000e-04
Epoch 96/1000
10/10 [==============================] - 1s 85ms/step - loss: 0.7413 - accuracy:
0.7705 - val_loss: 2.0722 - val_accuracy: 0.5253 - lr: 5.0000e-04
Epoch 97/1000
10/10 [==============================] - 1s 78ms/step - loss: 0.7405 - accuracy:
0.7784 - val_loss: 1.7349 - val_accuracy: 0.5730 - lr: 5.0000e-04
Epoch 98/1000
10/10 [==============================] - 1s 78ms/step - loss: 0.7504 - accuracy:
```

```
0.7716 - val_loss: 1.7426 - val_accuracy: 0.5615 - lr: 5.0000e-04
Epoch 99/1000
10/10 [==============================] - 1s 77ms/step - loss: 0.7214 - accuracy:
0.7792 - val_loss: 1.5014 - val_accuracy: 0.6237 - lr: 5.0000e-04
Epoch 100/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.7316 - accuracy:
0.7758 - val_loss: 2.2792 - val_accuracy: 0.4536 - lr: 5.0000e-04
Epoch 101/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.7449 - accuracy:
0.7723 - val_loss: 2.4033 - val_accuracy: 0.4681 - lr: 5.0000e-04
Epoch 102/1000
10/10 [==============================] - 1s 71ms/step - loss: 0.7553 - accuracy:
0.7726 - val_loss: 1.9451 - val_accuracy: 0.5304 - lr: 5.0000e-04
Epoch 103/1000
10/10 [==============================] - 1s 81ms/step - loss: 0.7327 - accuracy:
0.7765 - val_loss: 1.8414 - val_accuracy: 0.5745 - lr: 5.0000e-04
Epoch 104/1000
10/10 [==============================] - 1s 78ms/step - loss: 0.7146 - accuracy:
0.7814 - val_loss: 2.1518 - val_accuracy: 0.4867 - lr: 5.0000e-04
Epoch 105/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.6963 - accuracy:
0.7880 - val_loss: 1.4173 - val_accuracy: 0.6563 - lr: 5.0000e-04
Epoch 106/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.7051 - accuracy:
0.7851 - val_loss: 1.9284 - val_accuracy: 0.5354 - lr: 5.0000e-04
Epoch 107/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.6963 - accuracy:
0.7842 - val_loss: 1.8459 - val_accuracy: 0.5840 - lr: 5.0000e-04
Epoch 108/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.7077 - accuracy:
0.7801 - val_loss: 1.6991 - val_accuracy: 0.5911 - lr: 5.0000e-04
Epoch 109/1000
10/10 [==============================] - 1s 76ms/step - loss: 0.6903 - accuracy:
0.7844 - val_loss: 1.6777 - val_accuracy: 0.5911 - lr: 5.0000e-04
Epoch 110/1000
10/10 [==============================] - 1s 77ms/step - loss: 0.6972 - accuracy:
0.7829 - val_loss: 1.5008 - val_accuracy: 0.6197 - lr: 5.0000e-04
Epoch 111/1000
10/10 [==============================] - 1s 85ms/step - loss: 0.7162 - accuracy:
0.7814 - val_loss: 3.5267 - val_accuracy: 0.3879 - lr: 5.0000e-04
Epoch 112/1000
10/10 [==============================] - 1s 84ms/step - loss: 0.7030 - accuracy:
0.7835 - val_loss: 2.3600 - val_accuracy: 0.4812 - lr: 5.0000e-04
Epoch 113/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.6701 - accuracy:
0.7946 - val_loss: 1.4935 - val_accuracy: 0.6523 - lr: 5.0000e-04
Epoch 114/1000
10/10 [==============================] - 1s 77ms/step - loss: 0.6797 - accuracy:
```

```
0.7929 - val_loss: 2.7850 - val_accuracy: 0.4250 - lr: 5.0000e-04
Epoch 115/1000
10/10 [==============================] - 1s 76ms/step - loss: 0.6739 - accuracy:
0.7941 - val_loss: 2.2523 - val_accuracy: 0.4867 - lr: 5.0000e-04
Epoch 116/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.6868 - accuracy:
0.7924 - val_loss: 2.0229 - val_accuracy: 0.5710 - lr: 5.0000e-04
Epoch 117/1000
10/10 [==============================] - 1s 79ms/step - loss: 0.7816 - accuracy:
0.7646 - val_loss: 3.8095 - val_accuracy: 0.3823 - lr: 5.0000e-04
Epoch 118/1000
10/10 [==============================] - 1s 89ms/step - loss: 0.7375 - accuracy:
0.7767 - val_loss: 2.5434 - val_accuracy: 0.4501 - lr: 5.0000e-04
Epoch 119/1000
10/10 [==============================] - 1s 80ms/step - loss: 0.6743 - accuracy:
0.7926 - val_loss: 1.7208 - val_accuracy: 0.5850 - lr: 5.0000e-04
Epoch 120/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.6556 - accuracy:
0.7991 - val_loss: 1.8456 - val_accuracy: 0.5524 - lr: 5.0000e-04
Epoch 121/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.6457 - accuracy:
0.8044 - val_loss: 1.6371 - val_accuracy: 0.6152 - lr: 5.0000e-04
Epoch 122/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.6611 - accuracy:
0.8003 - val_loss: 2.4097 - val_accuracy: 0.4676 - lr: 5.0000e-04
Epoch 123/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.6436 - accuracy:
0.8029 - val_loss: 1.6785 - val_accuracy: 0.6036 - lr: 5.0000e-04
Epoch 124/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.6180 - accuracy:
0.8114 - val_loss: 1.6146 - val_accuracy: 0.6061 - lr: 5.0000e-04
Epoch 125/1000
10/10 [==============================] - 1s 80ms/step - loss: 0.6488 - accuracy:
0.7995 - val_loss: 2.8770 - val_accuracy: 0.4240 - lr: 5.0000e-04
Epoch 126/1000
10/10 [==============================] - 1s 79ms/step - loss: 0.7179 - accuracy:
0.7760 - val_loss: 2.5048 - val_accuracy: 0.4431 - lr: 5.0000e-04
Epoch 127/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.7209 - accuracy:
0.7797 - val_loss: 3.1209 - val_accuracy: 0.3909 - lr: 5.0000e-04
Epoch 128/1000
10/10 [==============================] - 1s 71ms/step - loss: 0.6588 - accuracy:
0.7978 - val_loss: 1.7914 - val_accuracy: 0.5936 - lr: 5.0000e-04
Epoch 129/1000
10/10 [==============================] - 1s 77ms/step - loss: 0.6358 - accuracy:
0.8049 - val_loss: 1.7354 - val_accuracy: 0.5896 - lr: 5.0000e-04
Epoch 130/1000
10/10 [==============================] - ETA: 0s - loss: 0.6317 - accuracy:
```

```
0.8068
Epoch 130: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
10/10 [==============================] - 1s 71ms/step - loss: 0.6317 - accuracy:
0.8068 - val_loss: 1.7700 - val_accuracy: 0.6046 - lr: 5.0000e-04
Epoch 131/1000
10/10 [==============================] - 1s 79ms/step - loss: 0.5887 - accuracy:
0.8224 - val_loss: 1.5718 - val_accuracy: 0.6006 - lr: 2.5000e-04
Epoch 132/1000
10/10 [==============================] - 1s 79ms/step - loss: 0.5825 - accuracy:
0.8252 - val_loss: 1.4502 - val_accuracy: 0.6548 - lr: 2.5000e-04
Epoch 133/1000
10/10 [==============================] - 1s 80ms/step - loss: 0.5626 - accuracy:
0.8254 - val_loss: 1.3111 - val_accuracy: 0.7030 - lr: 2.5000e-04
Epoch 134/1000
10/10 [==============================] - 1s 79ms/step - loss: 0.5627 - accuracy:
0.8293 - val_loss: 1.6107 - val_accuracy: 0.6152 - lr: 2.5000e-04
Epoch 135/1000
10/10 [==============================] - 1s 78ms/step - loss: 0.5610 - accuracy:
0.8294 - val_loss: 1.5372 - val_accuracy: 0.6543 - lr: 2.5000e-04
Epoch 136/1000
10/10 [==============================] - 1s 77ms/step - loss: 0.5613 - accuracy:
0.8270 - val_loss: 1.3902 - val_accuracy: 0.6839 - lr: 2.5000e-04
Epoch 137/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.5749 - accuracy:
0.8222 - val_loss: 1.4410 - val_accuracy: 0.6729 - lr: 2.5000e-04
Epoch 138/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.5633 - accuracy:
0.8297 - val_loss: 1.4287 - val_accuracy: 0.6959 - lr: 2.5000e-04
Epoch 139/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.5405 - accuracy:
0.8369 - val_loss: 1.4379 - val_accuracy: 0.6864 - lr: 2.5000e-04
Epoch 140/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.5360 - accuracy:
0.8380 - val_loss: 1.4818 - val_accuracy: 0.6713 - lr: 2.5000e-04
Epoch 141/1000
10/10 [==============================] - 1s 80ms/step - loss: 0.5463 - accuracy:
0.8308 - val_loss: 1.6933 - val_accuracy: 0.6272 - lr: 2.5000e-04
Epoch 142/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.5367 - accuracy:
0.8342 - val_loss: 1.8863 - val_accuracy: 0.5921 - lr: 2.5000e-04
Epoch 143/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.5504 - accuracy:
0.8336 - val_loss: 1.2888 - val_accuracy: 0.7195 - lr: 2.5000e-04
Epoch 144/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.5369 - accuracy:
0.8327 - val_loss: 1.8144 - val_accuracy: 0.6187 - lr: 2.5000e-04
Epoch 145/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.5308 - accuracy:
```

```
0.8368 - val_loss: 1.2976 - val_accuracy: 0.7045 - lr: 2.5000e-04
Epoch 146/1000
10/10 [==============================] - 1s 86ms/step - loss: 0.5347 - accuracy:
0.8309 - val_loss: 1.6727 - val_accuracy: 0.6372 - lr: 2.5000e-04
Epoch 147/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.5256 - accuracy:
0.8436 - val_loss: 1.5105 - val_accuracy: 0.6834 - lr: 2.5000e-04
Epoch 148/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.5024 - accuracy:
0.8477 - val_loss: 1.4269 - val_accuracy: 0.7015 - lr: 2.5000e-04
Epoch 149/1000
10/10 [==============================] - 1s 76ms/step - loss: 0.5164 - accuracy:
0.8417 - val_loss: 1.5292 - val_accuracy: 0.6618 - lr: 2.5000e-04
Epoch 150/1000
10/10 [==============================] - 1s 89ms/step - loss: 0.5245 - accuracy:
0.8402 - val_loss: 2.0989 - val_accuracy: 0.5815 - lr: 2.5000e-04
Epoch 151/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.5173 - accuracy:
0.8395 - val_loss: 1.8333 - val_accuracy: 0.6267 - lr: 2.5000e-04
Epoch 152/1000
10/10 [==============================] - 1s 71ms/step - loss: 0.5386 - accuracy:
0.8330 - val_loss: 2.3368 - val_accuracy: 0.5263 - lr: 2.5000e-04
Epoch 153/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.5326 - accuracy:
0.8335 - val_loss: 2.5298 - val_accuracy: 0.5309 - lr: 2.5000e-04
Epoch 154/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.5146 - accuracy:
0.8373 - val_loss: 1.2990 - val_accuracy: 0.7175 - lr: 2.5000e-04
Epoch 155/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.5184 - accuracy:
0.8393 - val_loss: 1.4652 - val_accuracy: 0.6949 - lr: 2.5000e-04
Epoch 156/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.5005 - accuracy:
0.8477 - val_loss: 1.3322 - val_accuracy: 0.7180 - lr: 2.5000e-04
Epoch 157/1000
10/10 [==============================] - 1s 82ms/step - loss: 0.5113 - accuracy:
0.8389 - val_loss: 1.6873 - val_accuracy: 0.6463 - lr: 2.5000e-04
Epoch 158/1000
10/10 [==============================] - 1s 78ms/step - loss: 0.5225 - accuracy:
0.8394 - val_loss: 2.0717 - val_accuracy: 0.5750 - lr: 2.5000e-04
Epoch 159/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.5362 - accuracy:
0.8321 - val_loss: 1.7989 - val_accuracy: 0.6247 - lr: 2.5000e-04
Epoch 160/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.5253 - accuracy:
0.8386 - val_loss: 1.4313 - val_accuracy: 0.6994 - lr: 2.5000e-04
Epoch 161/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.5117 - accuracy:
```

```
0.8389 - val_loss: 1.4663 - val_accuracy: 0.6994 - lr: 2.5000e-04
Epoch 162/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.5022 - accuracy:
0.8466 - val_loss: 1.3695 - val_accuracy: 0.7210 - lr: 2.5000e-04
Epoch 163/1000
10/10 [==============================] - 1s 81ms/step - loss: 0.4998 - accuracy:
0.8468 - val_loss: 1.8780 - val_accuracy: 0.6116 - lr: 2.5000e-04
Epoch 164/1000
10/10 [==============================] - 1s 76ms/step - loss: 0.4942 - accuracy:
0.8496 - val_loss: 1.5349 - val_accuracy: 0.6663 - lr: 2.5000e-04
Epoch 165/1000
10/10 [==============================] - 1s 90ms/step - loss: 0.4979 - accuracy:
0.8414 - val_loss: 1.3911 - val_accuracy: 0.7035 - lr: 2.5000e-04
Epoch 166/1000
10/10 [==============================] - 1s 79ms/step - loss: 0.5081 - accuracy:
0.8446 - val_loss: 1.8783 - val_accuracy: 0.6402 - lr: 2.5000e-04
Epoch 167/1000
10/10 [==============================] - 1s 76ms/step - loss: 0.5085 - accuracy:
0.8416 - val_loss: 2.3452 - val_accuracy: 0.5263 - lr: 2.5000e-04
Epoch 168/1000
10/10 [==============================] - ETA: 0s - loss: 0.5163 - accuracy:
0.8437
Epoch 168: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
10/10 [==============================] - 1s 74ms/step - loss: 0.5163 - accuracy:
0.8437 - val_loss: 2.1829 - val_accuracy: 0.5886 - lr: 2.5000e-04
Epoch 169/1000
10/10 [==============================] - 1s 70ms/step - loss: 0.4919 - accuracy:
0.8492 - val_loss: 1.3812 - val_accuracy: 0.6924 - lr: 1.2500e-04
Epoch 170/1000
10/10 [==============================] - 1s 79ms/step - loss: 0.4962 - accuracy:
0.8429 - val_loss: 1.5853 - val_accuracy: 0.6984 - lr: 1.2500e-04
Epoch 171/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.4594 - accuracy:
0.8578 - val_loss: 1.4123 - val_accuracy: 0.7205 - lr: 1.2500e-04
Epoch 172/1000
10/10 [==============================] - 1s 84ms/step - loss: 0.4725 - accuracy:
0.8559 - val_loss: 1.3743 - val_accuracy: 0.7230 - lr: 1.2500e-04
Epoch 173/1000
10/10 [==============================] - 1s 76ms/step - loss: 0.4623 - accuracy:
0.8586 - val_loss: 1.3640 - val_accuracy: 0.7230 - lr: 1.2500e-04
Epoch 174/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.4772 - accuracy:
0.8559 - val_loss: 1.2681 - val_accuracy: 0.7336 - lr: 1.2500e-04
Epoch 175/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.4694 - accuracy:
0.8583 - val_loss: 1.4537 - val_accuracy: 0.6959 - lr: 1.2500e-04
Epoch 176/1000
10/10 [==============================] - 1s 76ms/step - loss: 0.4698 - accuracy:
```

```
0.8523 - val_loss: 1.2918 - val_accuracy: 0.7416 - lr: 1.2500e-04
Epoch 177/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.4581 - accuracy:
0.8585 - val_loss: 1.4493 - val_accuracy: 0.6954 - lr: 1.2500e-04
Epoch 178/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.4608 - accuracy:
0.8583 - val_loss: 1.3686 - val_accuracy: 0.7316 - lr: 1.2500e-04
Epoch 179/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.4652 - accuracy:
0.8552 - val_loss: 1.2498 - val_accuracy: 0.7456 - lr: 1.2500e-04
Epoch 180/1000
10/10 [==============================] - 1s 84ms/step - loss: 0.4523 - accuracy:
0.8631 - val_loss: 1.4519 - val_accuracy: 0.7075 - lr: 1.2500e-04
Epoch 181/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.4623 - accuracy:
0.8601 - val_loss: 1.4561 - val_accuracy: 0.7030 - lr: 1.2500e-04
Epoch 182/1000
10/10 [==============================] - 1s 71ms/step - loss: 0.4526 - accuracy:
0.8621 - val_loss: 1.3670 - val_accuracy: 0.7215 - lr: 1.2500e-04
Epoch 183/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.4512 - accuracy:
0.8624 - val_loss: 1.3968 - val_accuracy: 0.7100 - lr: 1.2500e-04
Epoch 184/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.4490 - accuracy:
0.8620 - val_loss: 1.3724 - val_accuracy: 0.7175 - lr: 1.2500e-04
Epoch 185/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.4572 - accuracy:
0.8574 - val_loss: 1.5897 - val_accuracy: 0.6739 - lr: 1.2500e-04
Epoch 186/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.4502 - accuracy:
0.8624 - val_loss: 1.4769 - val_accuracy: 0.7010 - lr: 1.2500e-04
Epoch 187/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.4568 - accuracy:
0.8597 - val_loss: 1.3072 - val_accuracy: 0.7466 - lr: 1.2500e-04
Epoch 188/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.4407 - accuracy:
0.8630 - val_loss: 1.5170 - val_accuracy: 0.6889 - lr: 1.2500e-04
Epoch 189/1000
10/10 [==============================] - 1s 84ms/step - loss: 0.4403 - accuracy:
0.8657 - val_loss: 1.2950 - val_accuracy: 0.7381 - lr: 1.2500e-04
Epoch 190/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.4422 - accuracy:
0.8633 - val_loss: 1.3444 - val_accuracy: 0.7301 - lr: 1.2500e-04
Epoch 191/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.4449 - accuracy:
0.8595 - val_loss: 1.3588 - val_accuracy: 0.7346 - lr: 1.2500e-04
Epoch 192/1000
10/10 [==============================] - 1s 76ms/step - loss: 0.4557 - accuracy:
```

```
0.8601 - val_loss: 1.4414 - val_accuracy: 0.7280 - lr: 1.2500e-04
Epoch 193/1000
10/10 [==============================] - 1s 77ms/step - loss: 0.4368 - accuracy:
0.8628 - val_loss: 1.4800 - val_accuracy: 0.7175 - lr: 1.2500e-04
Epoch 194/1000
10/10 [==============================] - 1s 70ms/step - loss: 0.4517 - accuracy:
0.8629 - val_loss: 1.4272 - val_accuracy: 0.7346 - lr: 1.2500e-04
Epoch 195/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.4349 - accuracy:
0.8631 - val_loss: 1.4113 - val_accuracy: 0.7291 - lr: 1.2500e-04
Epoch 196/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.4359 - accuracy:
0.8613 - val_loss: 1.7839 - val_accuracy: 0.6739 - lr: 1.2500e-04
Epoch 197/1000
10/10 [==============================] - 1s 85ms/step - loss: 0.4448 - accuracy:
0.8614 - val_loss: 1.6032 - val_accuracy: 0.7025 - lr: 1.2500e-04
Epoch 198/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.4253 - accuracy:
0.8672 - val_loss: 1.3494 - val_accuracy: 0.7321 - lr: 1.2500e-04
Epoch 199/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.4252 - accuracy:
0.8732 - val_loss: 1.4432 - val_accuracy: 0.7190 - lr: 1.2500e-04
Epoch 200/1000
10/10 [==============================] - 1s 79ms/step - loss: 0.4189 - accuracy:
0.8743 - val_loss: 1.4368 - val_accuracy: 0.7155 - lr: 1.2500e-04
Epoch 201/1000
10/10 [==============================] - 1s 71ms/step - loss: 0.4320 - accuracy:
0.8656 - val_loss: 1.3738 - val_accuracy: 0.7321 - lr: 1.2500e-04
Epoch 202/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.4320 - accuracy:
0.8682 - val_loss: 1.4616 - val_accuracy: 0.7110 - lr: 1.2500e-04
Epoch 203/1000
10/10 [==============================] - 1s 76ms/step - loss: 0.4208 - accuracy:
0.8756 - val_loss: 1.4906 - val_accuracy: 0.7155 - lr: 1.2500e-04
Epoch 204/1000
10/10 [==============================] - ETA: 0s - loss: 0.4306 - accuracy:
0.8676
Epoch 204: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
10/10 [==============================] - 1s 80ms/step - loss: 0.4306 - accuracy:
0.8676 - val_loss: 1.3672 - val_accuracy: 0.7301 - lr: 1.2500e-04
Epoch 205/1000
10/10 [==============================] - 1s 87ms/step - loss: 0.4199 - accuracy:
0.8684 - val_loss: 1.4129 - val_accuracy: 0.7275 - lr: 6.2500e-05
Epoch 206/1000
10/10 [==============================] - 1s 77ms/step - loss: 0.4129 - accuracy:
0.8732 - val_loss: 1.3314 - val_accuracy: 0.7386 - lr: 6.2500e-05
Epoch 207/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.4096 - accuracy:
```

```
0.8735 - val_loss: 1.3239 - val_accuracy: 0.7511 - lr: 6.2500e-05
Epoch 208/1000
10/10 [==============================] - 1s 70ms/step - loss: 0.4086 - accuracy:
0.8742 - val_loss: 1.3606 - val_accuracy: 0.7376 - lr: 6.2500e-05
Epoch 209/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.4130 - accuracy:
0.8719 - val_loss: 1.4158 - val_accuracy: 0.7306 - lr: 6.2500e-05
Epoch 210/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.4145 - accuracy:
0.8714 - val_loss: 1.2933 - val_accuracy: 0.7491 - lr: 6.2500e-05
Epoch 211/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.4068 - accuracy:
0.8745 - val_loss: 1.3167 - val_accuracy: 0.7466 - lr: 6.2500e-05
Epoch 212/1000
10/10 [==============================] - 1s 83ms/step - loss: 0.4046 - accuracy:
0.8762 - val_loss: 1.3867 - val_accuracy: 0.7341 - lr: 6.2500e-05
Epoch 213/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.4245 - accuracy:
0.8680 - val_loss: 1.3514 - val_accuracy: 0.7551 - lr: 6.2500e-05
Epoch 214/1000
10/10 [==============================] - 1s 77ms/step - loss: 0.4049 - accuracy:
0.8731 - val_loss: 1.3811 - val_accuracy: 0.7416 - lr: 6.2500e-05
Epoch 215/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.4071 - accuracy:
0.8726 - val_loss: 1.3973 - val_accuracy: 0.7296 - lr: 6.2500e-05
Epoch 216/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.4183 - accuracy:
0.8735 - val_loss: 1.4108 - val_accuracy: 0.7401 - lr: 6.2500e-05
Epoch 217/1000
10/10 [==============================] - 1s 76ms/step - loss: 0.4184 - accuracy:
0.8698 - val_loss: 1.4965 - val_accuracy: 0.7265 - lr: 6.2500e-05
Epoch 218/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.4084 - accuracy:
0.8761 - val_loss: 1.3957 - val_accuracy: 0.7301 - lr: 6.2500e-05
Epoch 219/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.3964 - accuracy:
0.8780 - val_loss: 1.3475 - val_accuracy: 0.7411 - lr: 6.2500e-05
Epoch 220/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.4115 - accuracy:
0.8698 - val_loss: 1.3514 - val_accuracy: 0.7386 - lr: 6.2500e-05
Epoch 221/1000
10/10 [==============================] - 1s 87ms/step - loss: 0.4121 - accuracy:
0.8743 - val_loss: 1.3512 - val_accuracy: 0.7401 - lr: 6.2500e-05
Epoch 222/1000
10/10 [==============================] - 1s 78ms/step - loss: 0.3977 - accuracy:
0.8795 - val_loss: 1.3618 - val_accuracy: 0.7321 - lr: 6.2500e-05
Epoch 223/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.4019 - accuracy:
```

```
0.8775 - val_loss: 1.3527 - val_accuracy: 0.7456 - lr: 6.2500e-05
Epoch 224/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.4009 - accuracy:
0.8786 - val_loss: 1.3443 - val_accuracy: 0.7431 - lr: 6.2500e-05
Epoch 225/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.3968 - accuracy:
0.8786 - val_loss: 1.3996 - val_accuracy: 0.7275 - lr: 6.2500e-05
Epoch 226/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.4110 - accuracy:
0.8704 - val_loss: 1.3261 - val_accuracy: 0.7451 - lr: 6.2500e-05
Epoch 227/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.4052 - accuracy:
0.8755 - val_loss: 1.3752 - val_accuracy: 0.7391 - lr: 6.2500e-05
Epoch 228/1000
10/10 [==============================] - 1s 87ms/step - loss: 0.4165 - accuracy:
0.8722 - val_loss: 1.3983 - val_accuracy: 0.7331 - lr: 6.2500e-05
Epoch 229/1000
10/10 [==============================] - ETA: 0s - loss: 0.4133 - accuracy:
0.8712
Epoch 229: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
10/10 [==============================] - 1s 73ms/step - loss: 0.4133 - accuracy:
0.8712 - val_loss: 1.3839 - val_accuracy: 0.7291 - lr: 6.2500e-05
Epoch 230/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.3955 - accuracy:
0.8804 - val_loss: 1.3327 - val_accuracy: 0.7396 - lr: 3.1250e-05
Epoch 231/1000
10/10 [==============================] - 1s 82ms/step - loss: 0.3877 - accuracy:
0.8782 - val_loss: 1.2840 - val_accuracy: 0.7471 - lr: 3.1250e-05
Epoch 232/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.3966 - accuracy:
0.8780 - val_loss: 1.3074 - val_accuracy: 0.7466 - lr: 3.1250e-05
Epoch 233/1000
10/10 [==============================] - 1s 71ms/step - loss: 0.3987 - accuracy:
0.8779 - val_loss: 1.3352 - val_accuracy: 0.7456 - lr: 3.1250e-05
Epoch 234/1000
10/10 [==============================] - 1s 77ms/step - loss: 0.3958 - accuracy:
0.8736 - val_loss: 1.3660 - val_accuracy: 0.7426 - lr: 3.1250e-05
Epoch 235/1000
10/10 [==============================] - 1s 83ms/step - loss: 0.4011 - accuracy:
0.8741 - val_loss: 1.3368 - val_accuracy: 0.7471 - lr: 3.1250e-05
Epoch 236/1000
10/10 [==============================] - 1s 77ms/step - loss: 0.3847 - accuracy:
0.8823 - val_loss: 1.3433 - val_accuracy: 0.7441 - lr: 3.1250e-05
Epoch 237/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.3936 - accuracy:
0.8796 - val_loss: 1.4280 - val_accuracy: 0.7376 - lr: 3.1250e-05
Epoch 238/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.3936 - accuracy:
```

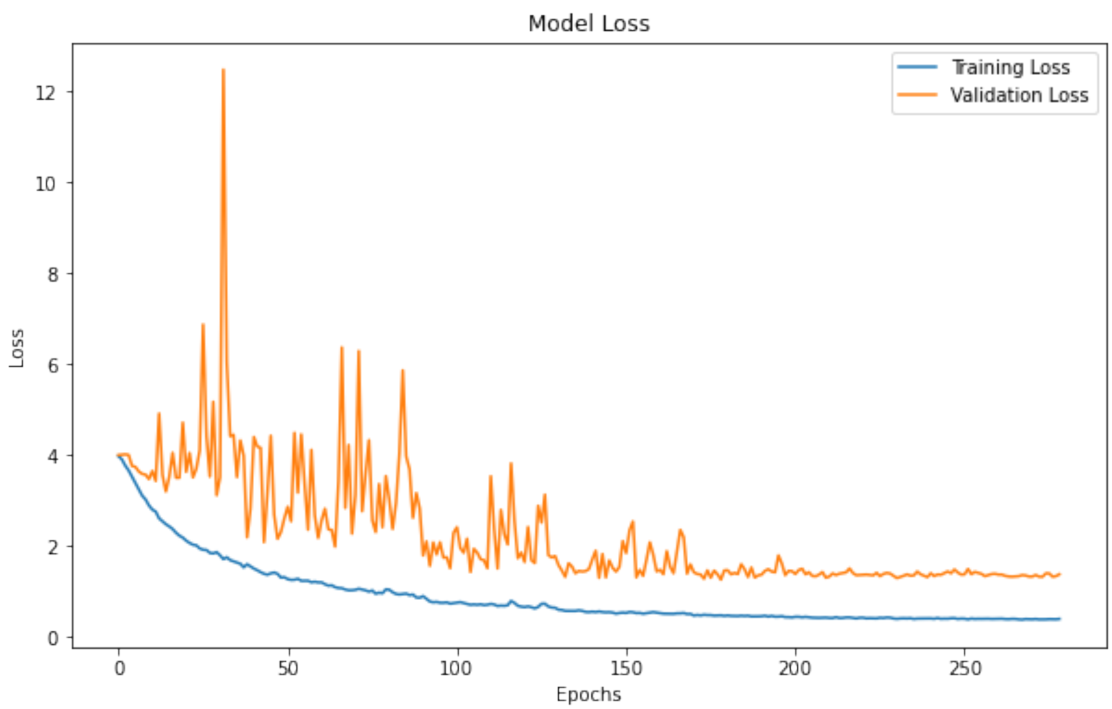
```
0.8783 - val_loss: 1.3677 - val_accuracy: 0.7436 - lr: 3.1250e-05
Epoch 239/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.3961 - accuracy:
0.8781 - val_loss: 1.3375 - val_accuracy: 0.7466 - lr: 3.1250e-05
Epoch 240/1000
10/10 [==============================] - 1s 76ms/step - loss: 0.3980 - accuracy:
0.8781 - val_loss: 1.3056 - val_accuracy: 0.7461 - lr: 3.1250e-05
Epoch 241/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.3969 - accuracy:
0.8757 - val_loss: 1.3906 - val_accuracy: 0.7331 - lr: 3.1250e-05
Epoch 242/1000
10/10 [==============================] - 1s 84ms/step - loss: 0.3882 - accuracy:
0.8817 - val_loss: 1.3279 - val_accuracy: 0.7421 - lr: 3.1250e-05
Epoch 243/1000
10/10 [==============================] - 1s 80ms/step - loss: 0.4055 - accuracy:
0.8771 - val_loss: 1.3637 - val_accuracy: 0.7351 - lr: 3.1250e-05
Epoch 244/1000
10/10 [==============================] - 1s 79ms/step - loss: 0.3882 - accuracy:
0.8812 - val_loss: 1.3565 - val_accuracy: 0.7416 - lr: 3.1250e-05
Epoch 245/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.3926 - accuracy:
0.8770 - val_loss: 1.3953 - val_accuracy: 0.7516 - lr: 3.1250e-05
Epoch 246/1000
10/10 [==============================] - 1s 69ms/step - loss: 0.3907 - accuracy:
0.8799 - val_loss: 1.4296 - val_accuracy: 0.7451 - lr: 3.1250e-05
Epoch 247/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.3999 - accuracy:
0.8755 - val_loss: 1.3914 - val_accuracy: 0.7456 - lr: 3.1250e-05
Epoch 248/1000
10/10 [==============================] - 1s 78ms/step - loss: 0.4023 - accuracy:
0.8785 - val_loss: 1.4735 - val_accuracy: 0.7255 - lr: 3.1250e-05
Epoch 249/1000
10/10 [==============================] - 1s 78ms/step - loss: 0.3850 - accuracy:
0.8798 - val_loss: 1.4053 - val_accuracy: 0.7386 - lr: 3.1250e-05
Epoch 250/1000
10/10 [==============================] - 1s 82ms/step - loss: 0.3932 - accuracy:
0.8789 - val_loss: 1.3613 - val_accuracy: 0.7406 - lr: 3.1250e-05
Epoch 251/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.3908 - accuracy:
0.8797 - val_loss: 1.3664 - val_accuracy: 0.7481 - lr: 3.1250e-05
Epoch 252/1000
10/10 [==============================] - 1s 71ms/step - loss: 0.3759 - accuracy:
0.8860 - val_loss: 1.4851 - val_accuracy: 0.7316 - lr: 3.1250e-05
Epoch 253/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.3945 - accuracy:
0.8770 - val_loss: 1.3693 - val_accuracy: 0.7436 - lr: 3.1250e-05
Epoch 254/1000
10/10 [==============================] - ETA: 0s - loss: 0.3848 - accuracy:
```

```
0.8823
Epoch 254: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
10/10 [==============================] - 1s 73ms/step - loss: 0.3848 - accuracy:
0.8823 - val_loss: 1.4202 - val_accuracy: 0.7265 - lr: 3.1250e-05
Epoch 255/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.3890 - accuracy:
0.8774 - val_loss: 1.3991 - val_accuracy: 0.7326 - lr: 1.5625e-05
Epoch 256/1000
10/10 [==============================] - 1s 81ms/step - loss: 0.3874 - accuracy:
0.8799 - val_loss: 1.3768 - val_accuracy: 0.7441 - lr: 1.5625e-05
Epoch 257/1000
10/10 [==============================] - 1s 76ms/step - loss: 0.3909 - accuracy:
0.8783 - val_loss: 1.3264 - val_accuracy: 0.7501 - lr: 1.5625e-05
Epoch 258/1000
10/10 [==============================] - 1s 83ms/step - loss: 0.3904 - accuracy:
0.8818 - val_loss: 1.3495 - val_accuracy: 0.7436 - lr: 1.5625e-05
Epoch 259/1000
10/10 [==============================] - 1s 77ms/step - loss: 0.3852 - accuracy:
0.8818 - val_loss: 1.3734 - val_accuracy: 0.7446 - lr: 1.5625e-05
Epoch 260/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.3858 - accuracy:
0.8799 - val_loss: 1.3787 - val_accuracy: 0.7451 - lr: 1.5625e-05
Epoch 261/1000
10/10 [==============================] - 1s 76ms/step - loss: 0.3836 - accuracy:
0.8825 - val_loss: 1.3607 - val_accuracy: 0.7511 - lr: 1.5625e-05
Epoch 262/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.3907 - accuracy:
0.8813 - val_loss: 1.3616 - val_accuracy: 0.7496 - lr: 1.5625e-05
Epoch 263/1000
10/10 [==============================] - 1s 71ms/step - loss: 0.3826 - accuracy:
0.8804 - val_loss: 1.3331 - val_accuracy: 0.7486 - lr: 1.5625e-05
Epoch 264/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.3802 - accuracy:
0.8849 - val_loss: 1.3200 - val_accuracy: 0.7481 - lr: 1.5625e-05
Epoch 265/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.3870 - accuracy:
0.8780 - val_loss: 1.3150 - val_accuracy: 0.7496 - lr: 1.5625e-05
Epoch 266/1000
10/10 [==============================] - 1s 81ms/step - loss: 0.3856 - accuracy:
0.8826 - val_loss: 1.3206 - val_accuracy: 0.7461 - lr: 1.5625e-05
Epoch 267/1000
10/10 [==============================] - 1s 76ms/step - loss: 0.3756 - accuracy:
0.8853 - val_loss: 1.3267 - val_accuracy: 0.7471 - lr: 1.5625e-05
Epoch 268/1000
10/10 [==============================] - 1s 69ms/step - loss: 0.3704 - accuracy:
0.8843 - val_loss: 1.3497 - val_accuracy: 0.7451 - lr: 1.5625e-05
Epoch 269/1000
10/10 [==============================] - 1s 70ms/step - loss: 0.3826 - accuracy:
```

```
0.8818 - val_loss: 1.3420 - val_accuracy: 0.7446 - lr: 1.5625e-05
Epoch 270/1000
10/10 [==============================] - 1s 75ms/step - loss: 0.3809 - accuracy:
0.8833 - val_loss: 1.3188 - val_accuracy: 0.7481 - lr: 1.5625e-05
Epoch 271/1000
10/10 [==============================] - 1s 77ms/step - loss: 0.3769 - accuracy:
0.8830 - val_loss: 1.3173 - val_accuracy: 0.7491 - lr: 1.5625e-05
Epoch 272/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.3838 - accuracy:
0.8824 - val_loss: 1.3520 - val_accuracy: 0.7446 - lr: 1.5625e-05
Epoch 273/1000
10/10 [==============================] - 1s 80ms/step - loss: 0.3746 - accuracy:
0.8824 - val_loss: 1.3159 - val_accuracy: 0.7501 - lr: 1.5625e-05
Epoch 274/1000
10/10 [==============================] - 1s 89ms/step - loss: 0.3733 - accuracy:
0.8864 - val_loss: 1.3117 - val_accuracy: 0.7551 - lr: 1.5625e-05
Epoch 275/1000
10/10 [==============================] - 1s 74ms/step - loss: 0.3757 - accuracy:
0.8853 - val_loss: 1.3879 - val_accuracy: 0.7451 - lr: 1.5625e-05
Epoch 276/1000
10/10 [==============================] - 1s 73ms/step - loss: 0.3810 - accuracy:
0.8802 - val_loss: 1.3871 - val_accuracy: 0.7436 - lr: 1.5625e-05
Epoch 277/1000
10/10 [==============================] - 1s 72ms/step - loss: 0.3806 - accuracy:
0.8835 - val_loss: 1.3131 - val_accuracy: 0.7556 - lr: 1.5625e-05
Epoch 278/1000
10/10 [==============================] - 1s 87ms/step - loss: 0.3776 - accuracy:
0.8838 - val_loss: 1.3273 - val_accuracy: 0.7521 - lr: 1.5625e-05
Epoch 279/1000
10/10 [==============================] - ETA: 0s - loss: 0.3857 - accuracy:
0.8814
Epoch 279: ReduceLROnPlateau reducing learning rate to 1e-05.
10/10 [==============================] - 1s 75ms/step - loss: 0.3857 - accuracy:
0.8814 - val_loss: 1.3669 - val_accuracy: 0.7401 - lr: 1.5625e-05
Epoch 279: early stopping
```

Gather metrics

[ ]:
```
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend()
plt.show()
```



[ ]:
```
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 10, 128)           916992

 dropout (Dropout)           (None, 10, 128)           0

 batch_normalization (Batch  (None, 10, 128)           512
 Normalization)

 lstm_1 (LSTM)               (None, 256)               394240

 dropout_1 (Dropout)         (None, 256)               0

 dense (Dense)               (None, 128)               32896

 dropout_2 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 53)                6837

=================================================================
Total params: 1351477 (5.16 MB)
```

```
Trainable params: 1351221 (5.15 MB)
Non-trainable params: 256 (1.00 KB)
_________________________________________________________________
```

Store the model data for later use in the Inference

```
[ ]: model_dir = 'model-' + MODEL_VERSION + '/'
model_name = 'model-' + MODEL_VERSION + '.h5'
model.save(model_dir + model_name)
labels_name = 'labels-' + MODEL_VERSION + '.npy'
np.save(model_dir + labels_name, actions)

df_csv = pd.DataFrame(actions)
labels_csv_name = 'labels-' + MODEL_VERSION + '.csv'
df_csv.to_csv(model_dir + labels_csv_name, index=False)
```

```
/home/user/.local/lib/python3.10/site-
packages/keras/src/engine/training.py:3079: UserWarning: You are saving your
model as an HDF5 file via `model.save()`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

```
[ ]: predictions = model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(y_test, axis=1)

accuracy = accuracy_score(true_classes, predicted_classes)
precision = precision_score(true_classes, predicted_classes, average='weighted')
recall = recall_score(true_classes, predicted_classes, average='weighted')
f1 = f1_score(true_classes, predicted_classes, average='weighted')
report = classification_report(true_classes, predicted_classes,␣
 ↪target_names=actions)

with open(model_dir + 'model_metrics.txt', 'w') as file:
    file.write(f"Accuracy: {accuracy}\n")
    file.write(f"Precision: {precision}\n")
    file.write(f"Recall: {recall}\n")
    file.write(f"F1 Score: {f1}\n")
    file.write(f"\nClassification Report:\n{report}")

print(report)
```

```
63/63 [==============================] - 1s 11ms/step
              precision    recall  f1-score   support

        aunt       0.92      0.83      0.87        41
        bird       0.85      0.72      0.78        46
       black       0.74      0.70      0.72        20
```

brother	0.61	0.67	0.64	21
brown	0.86	0.95	0.90	44
bug	0.87	0.82	0.85	40
callonphone	0.81	0.71	0.76	42
cheek	0.69	0.67	0.68	36
clown	0.84	0.84	0.84	61
cow	0.76	0.74	0.75	46
cute	0.90	0.76	0.83	34
dad	0.60	0.68	0.64	38
doll	0.78	0.58	0.67	43
donkey	0.61	0.79	0.69	34
drink	0.76	0.70	0.73	37
ear	0.83	0.89	0.86	44
eye	0.81	0.69	0.75	32
feet	0.82	0.85	0.84	39
find	0.86	0.76	0.81	33
fireman	0.64	0.81	0.71	37
flower	0.79	0.74	0.76	35
for	0.81	0.55	0.65	31
frog	0.83	0.78	0.81	37
grandpa	0.63	0.65	0.64	26
grass	0.74	0.72	0.73	32
gum	0.69	0.80	0.74	44
hair	0.85	0.79	0.81	42
hen	0.82	0.66	0.73	41
home	0.57	0.71	0.63	24
horse	0.81	0.68	0.74	50
lamp	0.53	0.55	0.54	31
mad	0.53	0.55	0.54	31
mom	0.66	0.66	0.66	35
mouse	0.66	0.71	0.68	58
nose	0.95	0.83	0.89	42
owl	0.86	0.80	0.83	46
pig	0.82	0.86	0.84	36
police	0.79	0.78	0.79	54
radio	0.78	0.86	0.82	44
see	0.76	0.93	0.84	41
shhh	0.78	0.90	0.83	48
shirt	0.63	0.94	0.76	18
sick	0.83	0.83	0.83	29
stairs	0.72	0.66	0.69	35
stuck	0.68	0.71	0.70	21
taste	0.78	0.86	0.82	37
thirsty	0.69	0.72	0.71	25
tiger	0.74	0.72	0.73	36
uncle	0.77	0.79	0.78	52
water	0.88	0.88	0.88	48
who	0.70	0.77	0.73	48

```
       yucky       0.80      0.64      0.71        25
       zebra       0.72      0.75      0.73        24

    accuracy                           0.76      1994
   macro avg       0.76      0.75      0.75      1994
weighted avg       0.77      0.76      0.76      1994
```

```
[ ]: if (save_high_performers):
         # Generate the classification report
         report = classification_report(true_classes, predicted_classes,␣
      ↪target_names=actions, output_dict=True)

         # Convert report to a DataFrame
         report_df = pd.DataFrame(report).transpose()

         # The last few rows are overall metrics (accuracy, macro avg, weighted␣
      ↪avg), so we remove them
         report_df = report_df[:-3]

         # Display the DataFrame sorted by F1-score
         report_df_sorted = report_df.sort_values(by='f1-score')

         high_performance_threshold = 0.82  # Define your threshold
         high_performing_categories = report_df_sorted[report_df_sorted['f1-score']␣
      ↪> high_performance_threshold]
         print(high_performing_categories.count())
         print(high_performing_categories)

         high_performing_labels = high_performing_categories.index.to_numpy()
         print(high_performing_labels)

         np.save(model_dir + 'high_performing_labels.npy', high_performing_labels)
```

```
precision    15
recall       15
f1-score     15
support      15
dtype: int64
       precision    recall  f1-score  support
taste   0.780488  0.864865  0.820513     37.0
cute    0.896552  0.764706  0.825397     34.0
sick    0.827586  0.827586  0.827586     29.0
owl     0.860465  0.804348  0.831461     46.0
shhh    0.781818  0.895833  0.834951     48.0
see     0.760000  0.926829  0.835165     41.0
feet    0.825000  0.846154  0.835443     39.0
clown   0.836066  0.836066  0.836066     61.0
```

```
pig      0.815789  0.861111  0.837838     36.0
bug      0.868421  0.825000  0.846154     40.0
ear      0.829787  0.886364  0.857143     44.0
aunt     0.918919  0.829268  0.871795     41.0
water    0.875000  0.875000  0.875000     48.0
nose     0.945946  0.833333  0.886076     42.0
brown    0.857143  0.954545  0.903226     44.0
['taste' 'cute' 'sick' 'owl' 'shhh' 'see' 'feet' 'clown' 'pig' 'bug' 'ear'
 'aunt' 'water' 'nose' 'brown']
```

[ ]:
```
if (save_low_performers):
    # Generate the classification report
    report = classification_report(true_classes, predicted_classes,␣
 ↪target_names=actions, output_dict=True)

    # Convert report to a DataFrame
    report_df = pd.DataFrame(report).transpose()

    # The last few rows are overall metrics (accuracy, macro avg, weighted␣
 ↪avg), so we remove them
    report_df = report_df[:-3]

    # Display the DataFrame sorted by F1-score
    report_df_sorted = report_df.sort_values(by='f1-score')

    low_performance_threshold = 0.6  # Define your threshold
    low_performing_categories = report_df_sorted[report_df_sorted['f1-score'] <␣
 ↪low_performance_threshold]
    print(low_performing_categories.count())
    print(low_performing_categories)

    low_performing_labels = low_performing_categories.index.to_numpy()
    print(low_performing_labels)

    np.save(model_dir + 'low_performing_labels.npy', low_performing_labels)
```

[ ]:
```
res = model.predict(X_test)
```

[ ]:
```
test_index = 10
print(actions[np.argmax(res[test_index])])
print(actions[np.argmax(y_test[test_index])])
```