הגדרת סביבת העבודה

- ייבוא ספריות חיוניות
- הפעלת אופטימיזציה (XLA ו־mixed precision)

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import random
import math

import tensorflow as tf
import tensorflow_hub as hub

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.layers import Dense, Activation, BatchNormalization, Dropout, GlobalAveragePooling2D, Glo
from tensorflow.keras.models import Model, Sequential, load_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.model_selection import train_test_split
```

```
# Fix random seeds to ensure reproducible results across runs
random.seed(42)          # Python’s built-in random module
np.random.seed(42)       # NumPy’s random number generator
tf.random.set_seed(42)   # TensorFlow’s random operations
```

```
# Show which GPUs TensorFlow can access
print("GPUs:", tf.config.list_physical_devices('GPU'))

# Turn on XLA (Accelerated Linear Algebra) JIT compilation for potential speedups
tf.config.optimizer.set_jit(True)

# Use mixed-precision training (float16) globally to accelerate on compatible GPUs
tf.keras.mixed_precision.set_global_policy('mixed_float16')
```

הכנת מערך הנתונים

- איסוף נתיבי קבצי התמונות והתוויות המתאימות
- ספירת תמונות לכל גזע ובדיקת שאין תיקיות ריקות
- יצירת DataFrame (של זוגות נתיב קובץ, תווית)

```
# Dataset dir; target shape = (224, 224, 3)
IMAGES_DIR = '/content/drive/Othercomputers/My Mac/Desktop/Stanford_Dogs/Images'
img_width, img_height = 224, 224
channels = 3
```

```
# Build lists of all image file paths and their labels; also record image count per breed
filepaths, labels = [], []
images_per_breed = {}

for folder in os.listdir(IMAGES_DIR):
    breed_path = os.path.join(IMAGES_DIR, folder)
    # Skip non-directory entries (e.g., hidden files)
    if not os.path.isdir(breed_path):
        continue

    # Folder names are like "n02102040-english_setter"; take text after the first dash as the label
    label = folder.split('-', 1)[1]
    # Count images in this breed’s folder for data inspection/validation
    images_per_breed[label] = len(os.listdir(breed_path))

    # Append each image’s full path and its corresponding label
    for img in os.listdir(breed_path):
        filepaths.append(os.path.join(breed_path, img))
        labels.append(label)
```

```
# check all labels
```

```python
assert len(images_per_breed) == 120, (f'There are {len(images_per_breed)}/120 breed folders.')
print('All breed folders are present!')

# check images count
no_images = dict(filter(lambda item: item[1] == 0, images_per_breed.items())) # dict only with images that have
assert len(no_images) == 0, (
    f'There are {len(no_images)}/120 empty breed folders. \nlabels of the missing images: {[item[0] for item in
    )
print('All images are in their correct folders!')
print("There is no missing data!")
```

In [ ]:
```python
# List all recorded breed labels (keys of images_per_breed)
list(images_per_breed.keys())
```

In [ ]:
```python
# Create a DataFrame pairing each image file path with its breed label
df = pd.DataFrame({'filepath': filepaths, 'label': labels})
```

In [ ]:
```python
# split the dataframe into train and test sets with a ration of 80/20
train_df, val_df = train_test_split(
    df,
    test_size=0.2,
    stratify=df['label'], # keep the disribution between train and test sets.
    random_state=42
)
```

In [ ]:
```python
# Build the class-name list and mapping
class_names    = sorted(train_df['label'].unique())
label_to_index = {name: idx for idx, name in enumerate(class_names)}

# Add an integer column for each split
train_df['label_idx'] = train_df['label'].map(label_to_index)
val_df  ['label_idx'] = val_df  ['label'].map(label_to_index)

# Number of classes
num_classes = len(class_names)   # should be 120
```

In [ ]:
```python
def make_dataset(df, batch_size, num_classes, training=True):
    # Get file paths and integer labels
    paths  = df['filepath'].values
    labels = df['label_idx'].values.astype('int32')

    # Build initial Dataset
    ds = tf.data.Dataset.from_tensor_slices((paths, labels))

    # Define load & preprocess function
    def _load_and_preprocess(path, label):
        # Read image from disk
        image = tf.io.read_file(path)
        image = tf.image.decode_jpeg(image, channels=3)
        image = tf.image.resize(image, [img_height, img_width]) / 255.0

        if training:
            image = tf.image.random_flip_left_right(image)
            image = tf.image.random_brightness(image, max_delta=0.1)
            image = tf.image.random_contrast(image, 0.9, 1.1)

        # Convert label to one-hot
        label = tf.one_hot(label, depth=num_classes)
        return image, label

    # Apply preprocessing in parallel
    ds = ds.map(_load_and_preprocess, num_parallel_calls=tf.data.AUTOTUNE)

    # (Optional) Shuffle if training
    if training:
        ds = ds.shuffle(buffer_size=1024)

    # Batch, cache in RAM, and prefetch for performance
    ds = ds.batch(batch_size)
    ds = ds.cache()
    ds = ds.prefetch(tf.data.AUTOTUNE)

    return ds
```

הגדרת המודל ובחירת רשת בסיס

- טעינת רשתות טרנספר מוכרות
- בניית והוספת ראש מיוחד
- אימון המודל ושוואת ארכיכטורות הבסיס השונות בעזרת לולאה

In [ ]:
```python
def build_base(network):
    # Load pretrained network (without final classifier) as feature extractor
    base = network(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```python
    base.trainable = False       # Freeze base layers during initial training
    return base

def build_head(base):
    # Add a global pooling layer to collapse spatial dimensions
    x = GlobalAveragePooling2D()(base.output)

    # First fully connected block
    x = Dense(512, kernel_regularizer=l2(1e-4))(x)  # 512 units with L2 regularization
    x = BatchNormalization()(x)                     # Normalize activations
    x = Activation('relu')(x)                       # Non-linear activation
    x = Dropout(0.5)(x)                             # Drop 50% to reduce overfitting

    # Second fully connected block
    x = Dense(256, kernel_regularizer=l2(1e-4))(x)  # 256 units with L2 regularization
    x = BatchNormalization()(x)                     # Normalize activations
    x = Activation('relu')(x)                       # Non-linear activation
    x = Dropout(0.5)(x)                             # Drop 50% to reduce overfitting

    # Final softmax layer for 120 dog breed classes
    head = Dense(120, activation='softmax')(x)
    return Model(inputs=base.input, outputs=head)
```

In [ ]:
```python
# Pretrained CNN architectures for transfer learning
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.xception import Xception
from tensorflow.keras.applications import InceptionResNetV2
```

In [ ]:
```python
lr = 1e-04
batch_size = 32
```

In [ ]:
```python
# Train and compare models using different pretrained bases
results = []
base_networks = {
    'Xception': Xception,
    'InceptionV3': InceptionV3,
    'InceptionResNetV2': InceptionResNetV2
}

for base_name, base_network in base_networks.items():
    tf.keras.backend.clear_session()  # Reset state between runs

    # Prepare training and validation datasets
    train_ds = make_dataset(train_df, batch_size, num_classes, training=True)
    val_ds   = make_dataset(val_df,   batch_size, num_classes, training=False)

    # Determine how many steps per epoch
    steps_per_epoch   = math.ceil(len(train_df) / batch_size)
    validation_steps  = math.ceil(len(val_df)   / batch_size)

    # Build the model: frozen base + custom head
    model = build_head(build_base(base_network))

    # Compile with Adam & smoothed categorical crossentropy
    model.compile(
        optimizer=Adam(learning_rate=lr),
        loss=CategoricalCrossentropy(label_smoothing=0.05),
        metrics=['accuracy']
    )

    print(f'Training Network with Base: {base_name}')
    print('-' * 40)

    # Train for a fixed number of epochs
    history = model.fit(
        train_ds,
        epochs=10,
        steps_per_epoch=steps_per_epoch,
        validation_data=val_ds,
        validation_steps=validation_steps,
    )

    # Store each model's config, weights, and training history
    results.append({
        'base':    base_name,
        'weights': model.get_weights(),
        'config':  model.to_json(),
        'history': history.history
    })
```

In [ ]:
```python
def comp_dash(results):
    # Compile each model’s validation loss per epoch into a list of records
    records_loss = []
    for result in results:
        for ep, val_loss in enumerate(result['history']['val_loss'], start=1):
            records_loss.append({
                'Epoch': ep,
```

```
                'Validation Loss': val_loss,
                'Model': result['base']
            })
    # Create DataFrame for plotting
    df_loss = pd.DataFrame(records_loss)

    # Set plot style and initialize figure
    sns.set_style('darkgrid')
    fig, ax = plt.subplots(figsize=(9, 4), dpi=300)

    # Draw lineplot of validation loss over epochs for each model
    sns.lineplot(
        data=df_loss,
        x='Epoch', y='Validation Loss',
        hue='Model', palette='Set2',
        marker='o', ax=ax
    )
    ax.set_title('Validation Loss per Epoch')
    ax.legend(
        title='Model',
        fontsize='small',
        loc='upper right',
        bbox_to_anchor=(1.05, 1)
    )

    # Adjust layout and display the chart
    plt.tight_layout()
    plt.show()
```

In [ ]:
```
comp_dash(results)
```

המשך אימון הראש

- לקיחת הבסיס האופטימלי
- המשך אימון של ראש הרשת
- הוספת עצירה מוקדמת וצ׳קפוינט

In [ ]:
```
# Rebuild the InceptionResNetV2 model from saved JSON config and weights
from tensorflow.keras.models import model_from_json

# Find the result entry corresponding to InceptionResNetV2
entry = next(r for r in results if r['base'] == 'InceptionResNetV2')

# Load model architecture from JSON string
inceptionresnet = model_from_json(entry['config'])

# Assign the trained weights to the model
inceptionresnet.set_weights(entry['weights'])
```

In [ ]:
```
# add early-stopping to prevent overfit at high epoch:
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True,
    verbose=2
)

# Save best model weights:
checkpoint1 = ModelCheckpoint(
    'model_rs_results.weights.h5',
    monitor='val_loss',
    save_best_only=True,
    save_weights_only=True,
    verbose=1
)
```

In [ ]:
```
# Prepare training and validation datasets
train_ds = make_dataset(train_df, batch_size, num_classes, training=True)
val_ds   = make_dataset(val_df,   batch_size, num_classes, training=False)

# Calculate how many batches per epoch for both sets
steps_per_epoch   = math.ceil(len(train_df) / batch_size)
validation_steps  = math.ceil(len(val_df)   / batch_size)

# Fine-tune the loaded InceptionResNetV2 model
# – Train for up to 10 epochs
# – Use early stopping to halt on no validation improvement
# – Save best weights via checkpoint callback
history = inceptionresnet.fit(
    train_ds,
    epochs=10,
    steps_per_epoch=steps_per_epoch,
    validation_data=val_ds,
```

```
    validation_steps=validation_steps,
    callbacks=[early_stop, checkpoint]
)
```

In [ ]:
```
# Plot training vs. validation accuracy over epochs; mark when the model was saved and optionally save the figu
def plot_acc(history, save_epoch=None, save_path=None):
    sns.set_style('darkgrid')
    epochs = range(1, len(history['accuracy']) + 1)

    plt.figure(figsize=(9, 4), dpi=300)

    # Draw vertical line at the epoch where the model was saved
    if save_epoch:
        plt.axvline(x=save_epoch, linestyle='--', color='gray', alpha=0.6, label='Model Saved')

    # Plot training and validation accuracy
    sns.lineplot(x=epochs, y=history['accuracy'], label='Training', marker='o')
    sns.lineplot(x=epochs, y=history['val_accuracy'], label='Validation', marker='o')

    plt.title('Training vs. Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend(loc='lower right')
    plt.tight_layout()

    # Save the plot to file if a path is provided
    if save_path:
        plt.savefig(save_path)

    plt.show()
```

In [ ]:
```
# Merge original training history with fine-tuning history (run only once)
concat_hist = {k: [] for k in history.history.keys()}

for key in concat_hist:
    # Append metrics from the initial pre-fine-tuning run
    concat_hist[key].extend(entry['history'][key])

    # Append metrics from the subsequent fine-tuning run
    concat_hist[key].extend(history.history[key])
```

In [ ]:
```
plot_acc(concat_hist)
```

הפשרת הבסיס וכיוונון עדין של המודל

- טעינת המודל מהצ׳אקפוינט האחרון
- הוספת מיתון קצב למידה
- הפשרת 75 שכבות מהלמעלה של מודל הבסיס

In [ ]:
```
trainable_layers = 75
lr = 1e-6
batch_size = 64
epochs = 10
```

In [ ]:
```
prev_name = 'model_rs_results.weights.h5'
HOME_PATH = '/content/drive/MyDrive/Machine Learning Projects/Dog Breed classification project/'
```

In [ ]:
```
# Filename for saving the fine-tuned model, including number of trainable layers and LR
log_name = f'model_ft_results(layers={trainable_layers}, lr={lr}).keras'

# Stop training if val_loss doesn’t improve by ≥1e-4 for 4 epochs
early_stop2 = EarlyStopping(
    monitor='val_loss',
    patience=4,
    min_delta=1e-4,
    restore_best_weights=False,
    verbose=2
)

# Save the best model based on highest val_accuracy to HOME_PATH + log_name
checkpoint2 = ModelCheckpoint(
    HOME_PATH + log_name,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True,
    verbose=2
)

# Halve the learning rate if val_loss doesn’t improve for 2 epochs, down to at least 1e-7
lr_reduce_on_plateau = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=2,
```

```python
    min_lr=1e-7,
    verbose=2
)
```

In [ ]:
```python
# Prepare TF datasets for training & validation, then compute batch counts per epoch
train_ds = make_dataset(train_df, batch_size, num_classes, training=True)
val_ds   = make_dataset(val_df,   batch_size, num_classes, training=False)

# Number of steps (batches) per epoch for training and validation
steps_per_epoch    = math.ceil(len(train_df) / batch_size)
validation_steps   = math.ceil(len(val_df)   / batch_size)
```

In [ ]:
```python
# Fine-tune the top layers of InceptionResNetV2:
# – Rebuild model head+base and load pretrained weights
# – Unfreeze last `trainable_layers` for gradient updates
# – Recompile and train with early stopping & LR reduction
inceptionresnet = build_head(build_base(InceptionResNetV2))
inceptionresnet.load_weights(HOME_PATH + prev_name)

for layer in inceptionresnet.layers[-trainable_layers:]:
    layer.trainable = True

inceptionresnet.compile(
    optimizer=Adam(learning_rate=lr),
    loss=CategoricalCrossentropy(label_smoothing=0.05),
    metrics=['accuracy']
)

print(f'Unfrozen Layers (top) = {trainable_layers}')
print('-' * 30)

fine_tuning_history = inceptionresnet.fit(
    train_ds,
    epochs=epochs,
    steps_per_epoch=steps_per_epoch,
    validation_data=val_ds,
    validation_steps=validation_steps,
    callbacks=[early_stop, lr_reduce_on_plateau]
)
```

In [ ]:
```python
# Save the fine-tuned InceptionResNetV2 model (architecture + weights) to disk
inceptionresnet.save(HOME_PATH + log_name)
```

In [ ]:
```python
# Set filename & path for the accuracy plot; find the 1-based epoch with lowest validation accuracy
plot_name = f'training_val_accuracy(layers={trainable_layers}, lr={lr}).png'
plot_path = HOME_PATH + plot_name
save_epoch = np.argmin(fine_tuning_history.history['val_accuracy']) + 1
```

In [ ]:
```python
plot_acc(fine_tuning_history.history, save_epoch=10, save_path=plot_path)
```

מיצוי אפוקים אחרונים לפני שמירת המודל הסופי

- המשכת תהליך האימון עד לסופו
- שמירת המודל הסופי

In [ ]:
```python
# Initialize a merged history dict (run once!) and copy over existing fine-tuning metrics
concat_versions_hist = {k: [] for k in fine_tuning_history.history.keys()}
for key in concat_versions_hist:
    concat_versions_hist[key].extend(fine_tuning_history.history[key])
```

In [ ]:
```python
# Start model save version counter at 1
ver = 1
```

In [ ]:
```python
# Set new Hyper-parameters for Final Epochs
batch_size = 64
epochs = 4
lr = 1e-06
```

In [ ]:
```python
train_ds = make_dataset(train_df, batch_size, num_classes, training=True)
val_ds = make_dataset(val_df,   batch_size, num_classes, training=False)

steps_per_epoch = math.ceil(len(train_df) / batch_size)
validation_steps = math.ceil(len(val_df) / batch_size)
```

In [ ]:
```python
# Load the previously saved model and unfreeze its top layers for fine-tuning
inceptionresnet = load_model(HOME_PATH + log_name)
for layer in inceptionresnet.layers[-trainable_layers:]:
    layer.trainable = True

# Recompile with the chosen optimizer, loss, and metric
inceptionresnet.compile(
```

```
    optimizer=Adam(learning_rate=lr),
    loss=CategoricalCrossentropy(label_smoothing=0.05),
    metrics=['accuracy']
)

print(f'Unfrozen Layers (top) = {trainable_layers}')
print('-' * 30)

# Fine-tune the model on our data
fine_tuning_history = inceptionresnet.fit(
    train_ds,
    epochs=epochs,
    steps_per_epoch=steps_per_epoch,
    validation_data=val_ds,
    validation_steps=validation_steps,
    callbacks=[early_stop]
)

# Prompt to save the new version; if yes, bump version, merge histories, and save
ans = input('save this model [y/n]: ')
if ans.lower() == 'y':
    ver += 1
    for key in concat_versions_hist:
        concat_versions_hist[key].extend(fine_tuning_history.history[key])
    inceptionresnet.save(HOME_PATH + log_name + f'v{ver}')
    print('Saved: ' + log_name + f'v{ver}')
else:
    # Discard session if not saving
    tf.keras.backend.clear_session()
```

In [ ]:
```
# Path to save the combined accuracy plot (version 2)
perf_image_path = "/content/drive/MyDrive/Machine Learning Projects/Dog Breed classification project/model_ft_r

# Plot training vs. validation accuracy from concat_versions_hist, mark epoch 14, and save to perf_image_path
plot_acc(concat_versions_hist, save_epoch=14, save_path=perf_image_path)
```

In [ ]:
```
# Set filepath for the current version and save the fine-tuned model there
new_path = HOME_PATH + log_name + f'v{ver}'
inceptionresnet.save(new_path)
```

יצוא המודל הסופי לגרסא קלה יותר

- TFLite Convertor

In [ ]:
```
# Load a saved .keras model from disk (without recompiling)
model_path = "/content/drive/Othercomputers/My Mac/Desktop/Dog Classification Project/Deploy Model/model_ft_res
model = tf.keras.models.load_model(model_path, compile=False)
```

In [ ]:
```
# Initialize a TFLite converter from the loaded Keras model
converter = tf.lite.TFLiteConverter.from_keras_model(model)

# Enable both standard TFLite ops and TensorFlow fallback (Flex) ops
converter.target_spec.supported_ops = [
    tf.lite.OpsSet.TFLITE_BUILTINS,  # core TFLite operations
    tf.lite.OpsSet.SELECT_TF_OPS     # TensorFlow ops fallback
]
```

In [ ]:
```
# Convert the loaded Keras model to a TensorFlow Lite flatbuffer
tflite_model = converter.convert()

# Define where to save the TFLite file
save_dir  = "/content/drive/Othercomputers/My Mac/Desktop/Dog Classification Project/Deploy Model/"
model_name = "deploy_model.tflite"
file_path = os.path.join(save_dir, model_name)

# Write the TFLite model to disk
with open(file_path, "wb") as f:
    f.write(tflite_model)

print("Saved TFLite model")
```