```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import os
import re
import pathlib
import shutil
import random
import string

# --- Helper function to suppress TensorFlow C-level logs ---
# 0 = all logs (default), 1 = filter INFO, 2 = filter WARNING, 3 = filt
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

# === 1. Download and Prepare Data ===

def download_and_extract_imdb():
    """
    Downloads and extracts the ACL IMDB dataset.
    Returns the pathlib.Path to the base extracted directory, or None c
    """
    url = "https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar
    dataset_tar_gz = "aclImdb_v1.tar.gz"
    target_imdb_dir_name = "aclImdb"
    target_imdb_dir_path = pathlib.Path(target_imdb_dir_name)

    # --- Cleanup previous extractions/downloads to ensure a clean slat
    print("Cleaning up previous IMDB related files/directories if they
    if os.path.exists(target_imdb_dir_path):
        print(f"Removing existing '{target_imdb_dir_path}' directory...
        shutil.rmtree(target_imdb_dir_path)
    if os.path.exists(dataset_tar_gz):
        print(f"Removing existing '{dataset_tar_gz}' file...")
        os.remove(dataset_tar_gz)
    # A non-standard extracted dir name from user's previous log
    if os.path.exists("aclImdb_v1_extracted"):
        print("Removing existing 'aclImdb_v1_extracted' directory...")
        shutil.rmtree("aclImdb_v1_extracted")
    print("Cleanup complete.")
    # End Cleanup

    print("Downloading IMDB dataset...")
    # cache_subdir='' places extraction in the current directory
    # This utility returns the path to the *extracted directory*
    extracted_dataset_path_str = keras.utils.get_file(
        dataset_tar_gz, url,
        untar=True, cache_dir='.',
        cache_subdir='')
    # Use the path returned by the keras utility
    dataset_dir = pathlib.Path(extracted_dataset_path_str)

    if not os.path.exists(dataset_dir):
        print(f"Error: Keras utility did not extract to '{dataset_dir}'
        return None # Indicate failure

    # dataset_dir is likely 'aclImdb_v1_extracted'. We want it to be 'a
    if dataset_dir.name != target_imdb_dir_name:
        print(f"Renaming extracted directory from '{dataset_dir.name}'
        if os.path.exists(target_imdb_dir_path):
            shutil.rmtree(target_imdb_dir_path) # Remove 'aclImdb' if i
        shutil.move(str(dataset_dir), str(target_imdb_dir_path))
        dataset_dir = target_imdb_dir_path # Update our path to point t

    if not os.path.exists(dataset_dir):
         print(f"Error: Renamed directory '{dataset_dir}' not found.")
         return None

    # Handle nested 'aclImdb/aclImdb' directory
    nested_imdb_dir = dataset_dir / target_imdb_dir_name
    if os.path.exists(nested_imdb_dir) and nested_imdb_dir.is_dir():
        print(f"Detected nested directory structure. Using '{nested_imc
        dataset_dir = nested_imdb_dir

    # Remove the unsupervised training data
    unsup_dir = dataset_dir / "train" / "unsup"
    if os.path.exists(unsup_dir):
        print(f"Removing {unsup_dir}...")
        shutil.rmtree(unsup_dir)

    return dataset_dir # Return the actual base aclImdb path

def setup_custom_directories(base_imdb_dir, train_samples_count, val_sa
    """
    Sets up specific train/val/test directories based on assignment spe
    - train_dir_subset: Contains `train_samples_count` samples.
    - val_dir: Contains `val_samples_count` samples. (SKIPS if exists)
    - test_dir_main: Contains the remaining test samples. (SKIPS if exi
    """
    print(f"\nSetting up custom data directories (Train: {train_samples
    main_train_dir = base_imdb_dir / "train"
    main_test_dir = base_imdb_dir / "test"

    # --- Create Validation Set (10,000 samples) ---
    val_dir = base_imdb_dir / "val_set"
    test_dir_main = base_imdb_dir / "test_set" # The remaining test fil

    # Only create val_set and test_set if they don't already exist.
    if not os.path.exists(val_dir) or not os.path.exists(test_dir_main)
```

```
        print("Creating new Validation and Test sets...")
        # Ensure directories are clean
        if os.path.exists(val_dir): shutil.rmtree(val_dir)
        if os.path.exists(test_dir_main): shutil.rmtree(test_dir_main)

        os.makedirs(val_dir / "pos")
        os.makedirs(val_dir / "neg")
        os.makedirs(test_dir_main / "pos")
        os.makedirs(test_dir_main / "neg")

        val_per_cat = val_samples_count // 2

        # Move 5,000 pos and 5,000 neg from test -> val
        for category in ("pos", "neg"):
            test_files = os.listdir(main_test_dir / category)
            random.shuffle(test_files)
            # Copy to validation
            for fname in test_files[:val_per_cat]:
                shutil.copy(main_test_dir / category / fname, val_dir /
            # Copy remaining to new test set
            for fname in test_files[val_per_cat:]:
                shutil.copy(main_test_dir / category / fname, test_dir_

        print(f"Created validation set at '{val_dir}' ({val_samples_cou
        print(f"Created test set at '{test_dir_main}' (15,000 samples).
    else:
        print("Using existing Validation and Test sets.")

    # --- Create Training Subset Directory ---
    # This part should run every time to create the specific subset
    train_dir_subset = base_imdb_dir / f"train_subset_{train_samples_cc
    if os.path.exists(train_dir_subset):
        shutil.rmtree(train_dir_subset)

    os.makedirs(train_dir_subset / "pos")
    os.makedirs(train_dir_subset / "neg")

    samples_per_cat = train_samples_count // 2

    for category in ("pos", "neg"):
        train_files = os.listdir(main_train_dir / category)
        random.shuffle(train_files)
        for fname in train_files[:samples_per_cat]:
            shutil.copy(main_train_dir / category / fname, train_dir_su

    print(f"Created training subset at '{train_dir_subset}' ({train_sam
    return train_dir_subset, val_dir, test_dir_main
```

```
# === 2. Define Model Parameters and Text Vectorization ===
# Assignment Parameters
MAX_LENGTH = 150       # Cutoff reviews after 150 words
MAX_TOKENS = 10000    # Consider only top 10,000 words
INITIAL_TRAIN_SAMPLES = 100
VAL_SAMPLES = 10000
BATCH_SIZE = 32

def standardize_text(input_data):
    """
    Custom standardization function to lowercase, remove punctuation,
    As suggested by lecture slides (Slide 7).
    """
    lowercase = tf.strings.lower(input_data)
    stripped_html = tf.strings.regex_replace(lowercase, "<br />", " "
    return tf.strings.regex_replace(
        stripped_html, f"[{re.escape(string.punctuation)}]", ""
    )
def create_vectorization_layer(full_imdb_train_dir):
    """
    Creates and adapts a TextVectorization layer.
    IMPORTANT: We adapt on the *full* training set (20,000 samples)
    to build a meaningful vocabulary, even if we only train on 100 sa
    """
    print("\nAdapting TextVectorization layer on FULL training datase
    # Use the main training directory to adapt
    full_train_ds = keras.utils.text_dataset_from_directory(
        full_imdb_train_dir, batch_size=BATCH_SIZE
    )
    text_only_train_ds = full_train_ds.map(lambda x, y: x)

    text_vectorization = layers.TextVectorization(
        max_tokens=MAX_TOKENS,
        output_mode="int",
        output_sequence_length=MAX_LENGTH,
        standardize=standardize_text  # Use our custom standardizatio
    )
    text_vectorization.adapt(text_only_train_ds)
    print("Vectorization layer adapted.")
    return text_vectorization

def vectorize_datasets(vector_layer, train_dir, val_dir, test_dir):
    """
    Applies the vectorization layer to the train, val, and test datas
    """
    train_ds = keras.utils.text_dataset_from_directory(
        train_dir, batch_size=BATCH_SIZE
    )
    val_ds = keras.utils.text_dataset_from_directory(
        val_dir, batch_size=BATCH_SIZE
    )
    test_ds = keras.utils.text_dataset_from_directory(
        test_dir, batch_size=BATCH_SIZE
    )

    int_train_ds = train_ds.map(lambda x, y: (vector_layer(x), y), nu
    int_val_ds = val_ds.map(lambda x, y: (vector_layer(x), y), num_pa
    int_test_ds = test_ds.map(lambda x, y: (vector_layer(x), y), num_

    return int_train_ds, int_val_ds, int_test_ds
```

```
# === 3. Download Pre-trained Word Embeddings (GloVe) ===

def get_glove_embeddings(vector_layer):
    """
    Downloads GloVe embeddings and creates a pre-trained embedding ma
    """
    url = "http://nlp.stanford.edu/data/glove.6B.zip"
    print("\nDownloading GloVe embeddings...")
    glove_zip = keras.utils.get_file(
        "glove.6B.zip", url,
        extract=True, cache_dir='.',
        cache_subdir='') # Extracts to current dir

    glove_file_1 = pathlib.Path("glove.6B.100d.txt")
    glove_file_2 = pathlib.Path("glove.6B") / "glove.6B.100d.txt"

    if glove_file_1.exists():
        glove_file = glove_file_1
    elif glove_file_2.exists():
        glove_file = glove_file_2
    else:
        # Fallback to the 'glove_extracted' from your log
        glove_file_3 = pathlib.Path("glove_extracted") / "glove.6B.10
        if glove_file_3.exists():
            glove_file = glove_file_3
        else:
            print("Error: Could not find 'glove.6B.100d.txt'.")
            print(f"Checked: {glove_file_1}, {glove_file_2}, {glove_f
            return None

    print(f"Parsing {glove_file}...")

    embeddings_index = {}
    with open(glove_file, encoding='utf-8') as f:
        for line in f:
            word, coefs = line.split(maxsplit=1)
            coefs = np.fromstring(coefs, "f", sep=" ")
            embeddings_index[word] = coefs
    print(f"Found {len(embeddings_index)} word vectors.")

    embedding_dim = 100 # From glove.6B.100d.txt
    vocabulary = vector_layer.get_vocabulary()
    word_index = dict(zip(vocabulary, range(len(vocabulary))))

    embedding_matrix = np.zeros((MAX_TOKENS, embedding_dim))
    for word, i in word_index.items():
        if i >= MAX_TOKENS:
            continue
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector

    # Create the non-trainable Embedding layer
    glove_embedding_layer = layers.Embedding(
        MAX_TOKENS,
        embedding_dim,
        embeddings_initializer=keras.initializers.Constant(embedding_
        trainable=False, # Crucial! We don't train this layer.
        mask_zero=True  # Use masking
    )
    print("GloVe embedding layer created.")
    return glove_embedding_layer
```

```
# === 4. Define Model Architectures ===

def build_model_scratch():
    """
    Builds the RNN model with an embedding layer trained from scratch
    """
    inputs = keras.Input(shape=(None,), dtype="int64")
    # Embedding layer, 100 dimensions, trained from scratch
    x = layers.Embedding(MAX_TOKENS, 100, mask_zero=True)(inputs)
    x = layers.Bidirectional(layers.LSTM(32))(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(1, activation="sigmoid")(x)

    model = keras.Model(inputs, outputs)
    model.compile(optimizer="rmsprop",
                  loss="binary_crossentropy",
                  metrics=["accuracy"])
    return model

def build_model_pretrained(embedding_layer):
    """
    Builds the RNN model using the pre-trained GloVe embedding layer.
    """
    inputs = keras.Input(shape=(None,), dtype="int64")
    x = embedding_layer(inputs) # Use the passed pre-trained layer
    x = layers.Bidirectional(layers.LSTM(32))(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(1, activation="sigmoid")(x)

    model = keras.Model(inputs, outputs)
    model.compile(optimizer="rmsprop",
                  loss="binary_crossentropy",
                  metrics=["accuracy"])
    return model
```

```
# === 5. Training and Evaluation Function ===

def train_and_evaluate(model_builder, int_train_ds, int_val_ds, int_te
    """
    Helper function to compile, train, and evaluate a model.
    """
    print(f"\n--- Training Model: {model_name} ---")
    # Clear session to reset model state
    keras.backend.clear_session()

    model = model_builder(**builder_kwargs)
    # model.summary()

    # Early stopping to prevent overfitting
    callbacks = [
        keras.callbacks.EarlyStopping(
            monitor="val_accuracy",
            patience=5,
            restore_best_weights=True
        )
    ]

    history = model.fit(
        int_train_ds,
        validation_data=int_val_ds,
        epochs=20,
        callbacks=callbacks,
        verbose=1 # Set to 1 or 2 for more detail, 0 for quiet
    )

    best_val_acc = max(history.history['val_accuracy'])
    print(f"Best Validation Accuracy ({model_name}): {best_val_acc:.4

    # Evaluate on the final, unseen test set
    test_loss, test_acc = model.evaluate(int_test_ds, verbose=0)
    print(f"Final Test Accuracy ({model_name}): {test_acc:.4f}")
    return best_val_acc, test_acc
    return best_val_acc, test_acc
```

```
# === 6. Main Experiment ===

def run_experiment():
    """
    Executes the full assignment, comparing models and looping over sam
    """

    # --- Part 1: Initial Setup (100 Samples) ---
    base_imdb_dir = download_and_extract_imdb()
    if base_imdb_dir is None:
        print("Failed to setup IMDB directory. Exiting.")
        return

    train_subset_dir_100, val_dir, test_dir = setup_custom_directories(
        base_imdb_dir=base_imdb_dir,
        train_samples_count=INITIAL_TRAIN_SAMPLES,
        val_samples_count=VAL_SAMPLES
    )

    # Adapt vectorizer on main 'aclImdb/train' directory
    vector_layer = create_vectorization_layer(base_imdb_dir / "train")

    # Get vectorized datasets for the 100-sample experiment
    train_ds_100, val_ds_10k, test_ds_15k = vectorize_datasets(
        vector_layer, train_subset_dir_100, val_dir, test_dir
    )

    # Get pre-trained GloVe layer
    glove_layer = get_glove_embeddings(vector_layer)
    if glove_layer is None:
        print("Failed to load GloVe embeddings. Exiting.")
        return

    # --- Part 2: Run Comparison for 100 Samples ---
    print("\n\n=== EXPERIMENT 1: 100 Training Samples ===")

    val_acc_scratch_100, test_acc_scratch_100 = train_and_evaluate(
        build_model_scratch,
        train_ds_100, val_ds_10k, test_ds_15k,
        "Embedding (from Scratch)"
    )

    val_acc_glove_100, test_acc_glove_100 = train_and_evaluate(
        build_model_pretrained,
        train_ds_100, val_ds_10k, test_ds_15k,
        "Embedding (GloVe Pre-trained)",
        embedding_layer=glove_layer
    )
    print("\n--- EXPERIMENT 1 RESULTS ---")
    print(f"Test Accuracy (Scratch, 100 samples): {test_acc_scratch_100
    print(f"Test Accuracy (GloVe, 100 samples):   {test_acc_glove_100:.

    if test_acc_glove_100 > test_acc_scratch_100:
        print(">>> RESULT: Pre-trained GloVe model performed better.")
    else:
        print(">>> RESULT: 'From Scratch' model performed better.")


    # --- Part 3: Find Crossover Point ---
    print("\n\n=== EXPERIMENT 2: Finding Crossover Point ===")
    print("Testing different training sample sizes to see when 'from sc

    sample_counts = [100, 250, 500, 1000, 2000, 5000, 10000, 20000]

    for count in sample_counts:
        print(f"\n--- Testing with {count} samples ---")

        # Create the specific training subset
        # This call will *not* delete val_set or test_set anymore
        train_subset_dir, _, _ = setup_custom_directories(
            base_imdb_dir=base_imdb_dir,
            train_samples_count=count,
            val_samples_count=VAL_SAMPLES
        )

        # Vectorize just the training set (val/test are already loaded)
        # We reuse val_ds_10k and test_ds_15k from Part 1
        int_train_ds, _, _ = vectorize_datasets(vector_layer, train_sub

        # We only need validation accuracy for this comparison
        val_acc_scratch, _ = train_and_evaluate(
            build_model_scratch,
            int_train_ds, val_ds_10k, test_ds_15k,
            f"Scratch ({count} samples)"
        )

        val_acc_glove, _ = train_and_evaluate(
            build_model_pretrained,
            int_train_ds, val_ds_10k, test_ds_15k,
            f"GloVe ({count} samples)",
            embedding_layer=glove_layer
        )

        print(f"--- Summary for {count} samples ---")
        print(f"Val Acc (Scratch): {val_acc_scratch:.4f}")
        print(f"Val Acc (GloVe):   {val_acc_glove:.4f}")

        if val_acc_scratch > val_acc_glove:
            print(f"\n>>> CROSSOVER FOUND: At {count} samples, the 'fro
            print("   achieved a higher validation accuracy than the pr
            break
        elif count == sample_counts[-1]:
            print("\n>>> EXPERIMENT END: 'From scratch' model did not c
            print("   pre-trained GloVe model within the tested sample

if __name__ == "__main__":
    run_experiment()
```

```
Cleaning up previous IMDB related files/directories if they exist...
Cleanup complete.
Downloading IMDB dataset...
Downloading data from https://ai.stanford.edu/~amaas/data/sentiment/ac
84125825/84125825 ━━━━━━━━━━━━━━━━━━━━ 18s 0us/step
Renaming extracted directory from 'aclImdb_v1_extracted' to 'aclImdb'.
Detected nested directory structure. Using 'aclImdb/aclImdb'.
Removing aclImdb/aclImdb/train/unsup...

Setting up custom data directories (Train: 100)...
Creating new Validation and Test sets...
Created validation set at 'aclImdb/aclImdb/val_set' (10000 samples).
Created test set at 'aclImdb/aclImdb/test_set' (15,000 samples).
Created training subset at 'aclImdb/aclImdb/train_subset_100' (100 sam

Adapting TextVectorization layer on FULL training dataset...
Found 25000 files belonging to 2 classes.
Vectorization layer adapted.
Found 100 files belonging to 2 classes.
Found 10000 files belonging to 2 classes.
Found 15000 files belonging to 2 classes.

Downloading GloVe embeddings...
Downloading data from http://nlp.stanford.edu/data/glove.6B.zip
862182613/862182613 ━━━━━━━━━━━━━━━━━━━━ 161s 0us/step
Parsing glove_extracted/glove.6B.100d.txt...
Found 400000 word vectors.
GloVe embedding layer created.


=== EXPERIMENT 1: 100 Training Samples ===

--- Training Model: Embedding (from Scratch) ---
Epoch 1/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 20s 5s/step - accuracy: 0.4295 - loss: 0.6957
Epoch 2/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 22s 7s/step - accuracy: 0.6631 - loss: 0.6878
Epoch 3/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 14s 5s/step - accuracy: 0.7747 - loss: 0.6815
Epoch 4/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 27s 7s/step - accuracy: 0.7262 - loss: 0.6770
Epoch 5/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 21s 7s/step - accuracy: 0.8153 - loss: 0.6715
Epoch 6/20
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 21s 7s/step - accuracy: 0.8525 - loss: 0.6632
Epoch 7/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 21s 7s/step - accuracy: 0.9418 - loss: 0.6434
Epoch 8/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 14s 5s/step - accuracy: 0.9263 - loss: 0.6219
Epoch 9/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 21s 5s/step - accuracy: 0.7759 - loss: 0.5954
Epoch 10/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 20s 5s/step - accuracy: 0.9693 - loss: 0.5527
Epoch 11/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 14s 5s/step - accuracy: 0.9234 - loss: 0.4697
Epoch 12/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 17s 5s/step - accuracy: 0.8239 - loss: 0.4353
Epoch 13/20
```