cnn+transformer

July 22, 2025

[ ]:

```
import os
import cv2
import json

video_dir = r"C:\Users\Harsimran␣
 ↪Singh\Downloads\WLASL-master\start_kit\raw_videos"
output_dir = r"C:\Users\Harsimran␣
 ↪Singh\Downloads\WLASL-master\start_kit\wlasl_frames"
sequence_length = 16

with open(r"C:\Users\Harsimran Singh\Downloads\WLASL-master\start_kit\WLASL_v0.
 ↪3.json") as f:
    data = json.load(f)

for entry in data:
    label = entry["gloss"]
    for instance in entry["instances"]:
        video_id = instance["video_id"]
        video_path = os.path.join(video_dir, video_id + ".mp4")

        if not os.path.exists(video_path):
            continue

        cap = cv2.VideoCapture(video_path)
        frames = []
        while True:
            ret, frame = cap.read()
            if not ret:
                break
            frames.append(frame)
        cap.release()

        if len(frames) < sequence_length:
            continue

        # Sample SEQUENCE_LENGTH evenly spaced frames
        step = len(frames) // sequence_length
        sampled_frames = [frames[i] for i in range(0, len(frames), step)[:
 ↪sequence_length]]

        label_path = os.path.join(output_dir, label)
        os.makedirs(label_path, exist_ok=True)
        video_folder = os.path.join(label_path, video_id)
        os.makedirs(video_folder, exist_ok=True)

        for i, frame in enumerate(sampled_frames):
            out_path = os.path.join(video_folder, f"frame_{i:03d}.jpg")
            cv2.imwrite(out_path, frame)

print("  Frame extraction complete.")
```

[1]:

```
import os
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import (Input, TimeDistributed,␣
 ↪GlobalAveragePooling1D, Dropout, Dense, MultiHeadAttention, Add,␣
 ↪LayerNormalization, Embedding)
from tensorflow.keras.applications import EfficientNetV2B0
from tensorflow.keras.callbacks import EarlyStopping, LearningRateScheduler,␣
 ↪ModelCheckpoint
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from collections import Counter, deque
import json
from collections import Counter
```

[2]:

```
tf.random.set_seed(42)
np.random.seed(42)
```

[3]:

```
IMG_HEIGHT, IMG_WIDTH = 64, 64
SEQUENCE_LENGTH = 16
BATCH_SIZE = 16
EPOCHS = 40
```

[4]:

```
def augment_frame(img):
    if np.random.rand() < 0.3:
        img = cv2.flip(img, 1)
    if np.random.rand() < 0.3:
        img = cv2.GaussianBlur(img, (3,3), 0)
    if np.random.rand() < 0.3:
        hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        hsv[...,1] = hsv[...,1] * (0.8 + np.random.rand() * 0.4)
        img = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)
    if np.random.rand() < 0.3:
        brightness = 0.7 + np.random.rand() * 0.6
        img = np.clip(img * brightness, 0, 255).astype(np.uint8)
    return img
```

[5]:

```
def load_dataset(path):
    X, y = [], []
    label_list = sorted(os.listdir(path))
    label_map = {label: idx for idx, label in enumerate(label_list)}

    for label in label_list:
        label_path = os.path.join(path, label)
        for sample_folder in os.listdir(label_path):
            sample_path = os.path.join(label_path, sample_folder)
            frames = []
            frame_files = sorted(os.listdir(sample_path))[:SEQUENCE_LENGTH]
            for frame_name in frame_files:
                frame = cv2.imread(os.path.join(sample_path, frame_name))
                frame = cv2.resize(frame, (IMG_WIDTH, IMG_HEIGHT))
                frame = augment_frame(frame)
                frames.append(frame)
            if len(frames) == SEQUENCE_LENGTH:
                X.append(frames)
                y.append(label_map[label])

    class_counts = Counter(y)
    valid_classes = {cls for cls, count in class_counts.items() if count >= 2}
    X_filtered, y_filtered = [], []
    for xi, yi in zip(X, y):
        if yi in valid_classes:
            X_filtered.append(xi)
            y_filtered.append(yi)

    X = np.array(X_filtered)
    X = X.astype(np.float32)
    mean = np.mean(X)
    std = np.std(X)
    X = (X - mean) / (std + 1e-7)

    print(f" Filtered dataset: {len(valid_classes)} classes retained.")
    return X, to_categorical(y_filtered), label_map
```

[6]:

```
def transformer_encoder(inputs, head_size=64, num_heads=4, ff_dim=128,␣
 ↪dropout=0.2):
    x = MultiHeadAttention(num_heads=num_heads, key_dim=head_size)(inputs,␣
 ↪inputs)
```

```
    x = Dropout(dropout)(x)
    x = Add()([x, inputs])
    x = LayerNormalization()(x)

    ff = Dense(ff_dim, activation='relu')(x)
    ff = Dropout(dropout)(ff)
    ff = Dense(inputs.shape[-1])(ff)
    x = Add()([x, ff])
    return LayerNormalization()(x)
```

[7]:
```
def add_learned_positional_encoding(x):
    pos = tf.range(start=0, limit=SEQUENCE_LENGTH, delta=1)
    pos_emb = Embedding(input_dim=SEQUENCE_LENGTH, output_dim=x.shape[-1])(pos)
    return x + pos_emb
```

[8]:
```
def build_model(input_shape, num_classes):
    inputs = Input(shape=input_shape)

    base_cnn = EfficientNetV2B0(include_top=False, input_shape=(IMG_HEIGHT,␣
  ↪IMG_WIDTH, 3), pooling='avg', weights='imagenet')
    base_cnn.trainable = True
    x = TimeDistributed(base_cnn)(inputs)
    x = Dropout(0.2)(x)

    x = add_learned_positional_encoding(x)

    for _ in range(4):
        x = transformer_encoder(x, head_size=128, num_heads=4, ff_dim=256)

    x = GlobalAveragePooling1D()(x)
    x = Dropout(0.4)(x)
    x = Dense(128, activation='relu')(x)
    outputs = Dense(num_classes, activation='softmax')(x)

    return Model(inputs, outputs)
```

[9]:
```
X, y, label_map = load_dataset("C:/Users/Harsimran Singh/Downloads/WLASL-master/
  ↪start_kit/wlasl_frames")
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
  ↪stratify=y)

model = build_model((SEQUENCE_LENGTH, IMG_HEIGHT, IMG_WIDTH, 3), len(label_map))
model.compile(optimizer=tf.keras.optimizers.AdamW(learning_rate=1e-4,␣
  ↪weight_decay=1e-5),
              loss=tf.keras.losses.CategoricalCrossentropy(label_smoothing=0.
  ↪05),
              metrics=['accuracy', tf.keras.metrics.
 ↪TopKCategoricalAccuracy(k=3)])

def scheduler(epoch, lr):
    if epoch < 10:
        return lr
    else:
        return lr * tf.math.exp(-0.1)

callbacks = [
    EarlyStopping(patience=7, restore_best_weights=True),
    LearningRateScheduler(scheduler),
    ModelCheckpoint("best_model.h5", save_best_only=True,␣
 ↪monitor='val_accuracy', mode='max')
]

model.fit(X_train, y_train, validation_data=(X_val, y_val),
          epochs=EPOCHS, batch_size=BATCH_SIZE, callbacks=callbacks)
```

```
 Filtered dataset: 38 classes retained.
Epoch 1/40
17/17 [==============================] - 76s 3s/step - loss: 3.9139 - accuracy:
0.0338 - top_k_categorical_accuracy: 0.0977 - val_loss: 3.9003 - val_accuracy:
0.0149 - val_top_k_categorical_accuracy: 0.0597 - lr: 1.0000e-04
Epoch 2/40
17/17 [==============================] - 46s 3s/step - loss: 3.6212 - accuracy:
0.0602 - top_k_categorical_accuracy: 0.1203 - val_loss: 3.7045 - val_accuracy:
0.0299 - val_top_k_categorical_accuracy: 0.1343 - lr: 1.0000e-04
Epoch 3/40
17/17 [==============================] - 45s 3s/step - loss: 3.5807 - accuracy:
0.0639 - top_k_categorical_accuracy: 0.1692 - val_loss: 3.7474 - val_accuracy:
0.0299 - val_top_k_categorical_accuracy: 0.1343 - lr: 1.0000e-04
Epoch 4/40
17/17 [==============================] - 44s 3s/step - loss: 3.4726 - accuracy:
0.0865 - top_k_categorical_accuracy: 0.2105 - val_loss: 3.7414 - val_accuracy:
0.0597 - val_top_k_categorical_accuracy: 0.1045 - lr: 1.0000e-04
Epoch 5/40
17/17 [==============================] - 45s 3s/step - loss: 3.3517 - accuracy:
0.1316 - top_k_categorical_accuracy: 0.2857 - val_loss: 3.7624 - val_accuracy:
0.0597 - val_top_k_categorical_accuracy: 0.1194 - lr: 1.0000e-04
Epoch 6/40
17/17 [==============================] - 44s 3s/step - loss: 3.2097 - accuracy:
0.1805 - top_k_categorical_accuracy: 0.3233 - val_loss: 3.7869 - val_accuracy:
0.1343 - val_top_k_categorical_accuracy: 0.2239 - lr: 1.0000e-04
Epoch 7/40
17/17 [==============================] - 47s 3s/step - loss: 2.9285 - accuracy:
0.2331 - top_k_categorical_accuracy: 0.4436 - val_loss: 3.6938 - val_accuracy:
```

```
0.0896 - val_top_k_categorical_accuracy: 0.1940 - lr: 1.0000e-04
Epoch 8/40
17/17 [==============================] - 47s 3s/step - loss: 2.7941 - accuracy:
0.2519 - top_k_categorical_accuracy: 0.4812 - val_loss: 4.0741 - val_accuracy:
0.0746 - val_top_k_categorical_accuracy: 0.1791 - lr: 1.0000e-04
Epoch 9/40
17/17 [==============================] - 46s 3s/step - loss: 2.5061 - accuracy:
0.3722 - top_k_categorical_accuracy: 0.5526 - val_loss: 4.0608 - val_accuracy:
0.1791 - val_top_k_categorical_accuracy: 0.2239 - lr: 1.0000e-04
Epoch 10/40
17/17 [==============================] - 44s 3s/step - loss: 2.2426 - accuracy:
0.4737 - top_k_categorical_accuracy: 0.6353 - val_loss: 4.4786 - val_accuracy:
0.1045 - val_top_k_categorical_accuracy: 0.2239 - lr: 1.0000e-04
Epoch 11/40
17/17 [==============================] - 46s 3s/step - loss: 1.9017 - accuracy:
0.5526 - top_k_categorical_accuracy: 0.7707 - val_loss: 4.3516 - val_accuracy:
0.0597 - val_top_k_categorical_accuracy: 0.1045 - lr: 9.0484e-05
Epoch 12/40
17/17 [==============================] - 44s 3s/step - loss: 1.5229 - accuracy:
0.7218 - top_k_categorical_accuracy: 0.8872 - val_loss: 4.8396 - val_accuracy:
0.1493 - val_top_k_categorical_accuracy: 0.2388 - lr: 8.1873e-05
Epoch 13/40
17/17 [==============================] - 45s 3s/step - loss: 1.3486 - accuracy:
0.7406 - top_k_categorical_accuracy: 0.9135 - val_loss: 4.7247 - val_accuracy:
0.1642 - val_top_k_categorical_accuracy: 0.2537 - lr: 7.4082e-05
Epoch 14/40
17/17 [==============================] - 45s 3s/step - loss: 1.1743 - accuracy:
0.7744 - top_k_categorical_accuracy: 0.9286 - val_loss: 4.5940 - val_accuracy:
0.1045 - val_top_k_categorical_accuracy: 0.2090 - lr: 6.7032e-05
```

```
[9]: <keras.callbacks.History at 0x1e867a23310>
```

```
[12]: model.save(r"C:\Users\Harsimran Singh\Downloads\Untitled␣
       ↪Folder\cnn_transformer_sign_model.h5")
      with open("C:/Users/Harsimran Singh/Downloads/WLASL-master/start_kit/label_map.
       ↪json", "w") as f:
          json.dump({k: int(v) for k, v in label_map.items()}, f)

      print("Training complete and model saved.")
```

```
Training complete and model saved.
```

```
[11]: print("\n\ud83c\udfa5 Starting real-time prediction. Press 'q' to quit.")
      model = load_model("C:/Users/Harsimran Singh/Downloads/WLASL-master/start_kit/
       ↪cnn_transformer_sign_model.h5")
      with open("C:/Users/Harsimran Singh/Downloads/WLASL-master/start_kit/label_map.
       ↪json") as f:
          label_map = json.load(f)
```

```python
inv_label_map = {int(v): k for k, v in label_map.items()}

cap = cv2.VideoCapture(0)

def capture_sequence(cap, seq_len=SEQUENCE_LENGTH):
    frames = []
    for _ in range(seq_len):
        ret, frame = cap.read()
        if not ret:
            break
        frame = cv2.flip(frame, 1)
        resized = cv2.resize(frame, (IMG_WIDTH, IMG_HEIGHT))
        frames.append(resized.astype(np.float32))
    X_seq = np.array(frames)
    X_seq = (X_seq - np.mean(X_seq)) / (np.std(X_seq) + 1e-7)
    return X_seq

pred_queue = deque(maxlen=5)

while True:
    seq = capture_sequence(cap)
    if seq.shape[0] != SEQUENCE_LENGTH:
        continue

    input_tensor = np.expand_dims(seq, axis=0)
    preds = model.predict(input_tensor, verbose=0)
    label_idx = int(np.argmax(preds))
    pred_queue.append(label_idx)

    if len(pred_queue) == pred_queue.maxlen:
        most_common = Counter(pred_queue).most_common(1)[0][0]
        predicted_word = inv_label_map.get(most_common, "Unknown")
    else:
        predicted_word = inv_label_map.get(label_idx, "Unknown")

    for _ in range(5):
        ret, frame = cap.read()
        if not ret:
            break
        frame = cv2.putText(frame, f"Prediction: {predicted_word}", (20, 50),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        cv2.imshow("Sign Language Predictor", frame)
        if cv2.waitKey(30) & 0xFF == ord('q'):
            break

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

```

cap.release()
cv2.destroyAllWindows()
```