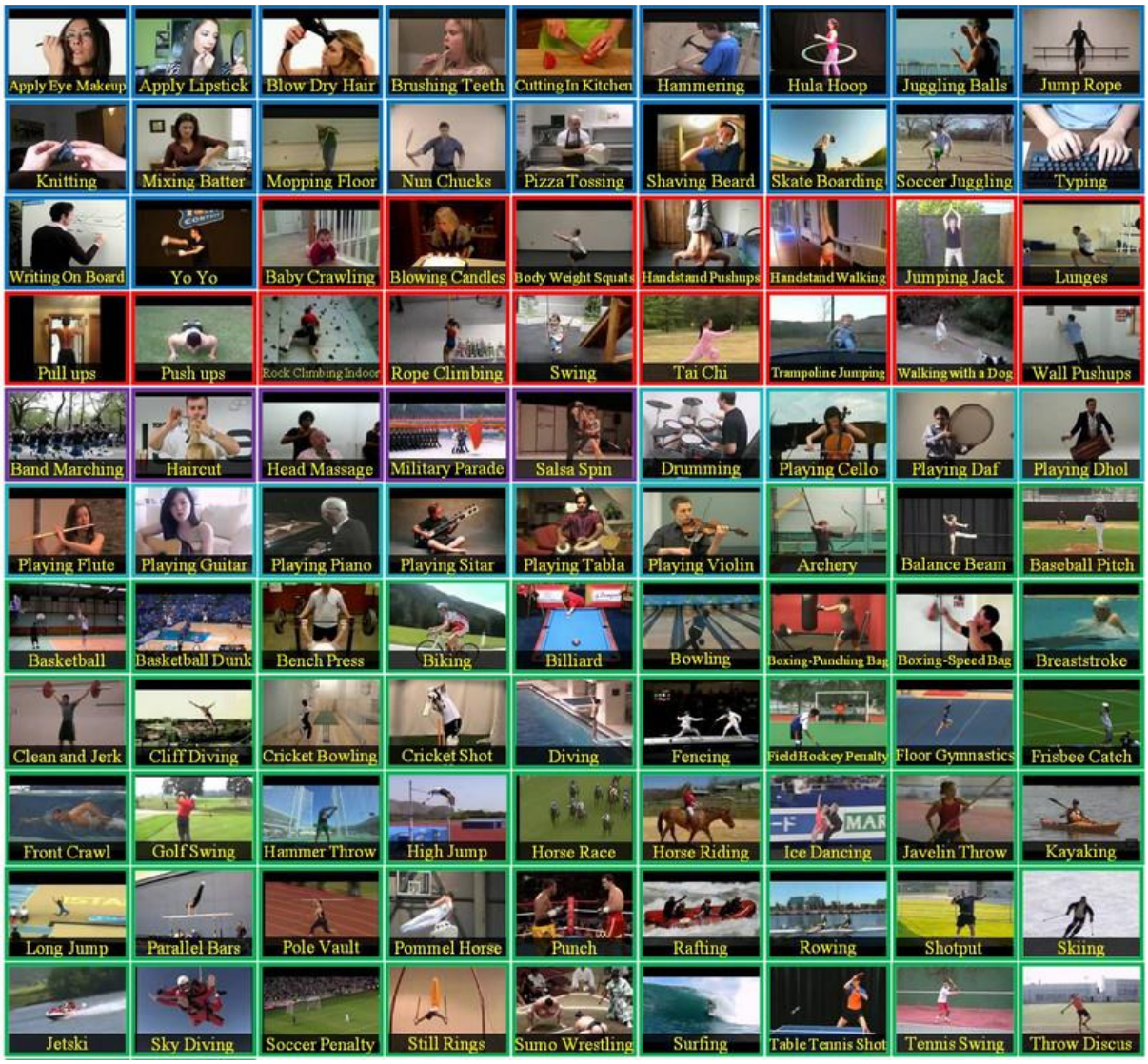
TP : Les Réseaux de Neurones Récurrents (RNNs)

Présentation de la base de données UCF101 :

La base de données *UCF101* est utilisée pour classifier des données vidéo en différentes catégories d'actions telles que "Basket-ball", "Playing Guitare" ou encore “Military Parade”.
L'objectif de ce TP est de classifier ce jeu de données en chargeant les vidéos et en entraînant un modèle de classification pour distinguer les 101 classes d'actions.



La reconnaissance d'action est une technique en vision par ordinateur qui consiste à identifier et à classer des actions ou des mouvements humains dans des vidéos ou des séquences d'images. Elle vise à analyser le contenu visuel d'une vidéo pour déterminer quelle action est en train d'être réalisée, comme courir, marcher, jouer au tennis, etc.

Cette méthodologie repose sur des modèles d'apprentissage automatique ou d'apprentissage profond, qui apprennent à partir de données annotées (comme des vidéos où chaque action est associée à une catégorie) afin de détecter des motifs récurrents dans les mouvements ou les changements de scène. Les applications courantes de la reconnaissance d'action incluent la surveillance vidéo, l'analyse de sports, la réalité augmentée et les interfaces homme-machine.

Vous utiliserez Jupyter Notebook en séparant le document en 4 parties :

- La première partie sera réservée aux bibliothèques que vous allez importer.
- La seconde sera destinée aux différentes variables.
- La troisième, pour les différentes fonctions que vous allez réaliser
- Et une dernière partie pour faire votre main.

Vous devez également télécharger la base de données UCF101 que vous pouvez trouver sur :
https://www.kaggle.com/datasets/abdallahwagih/ucf101-videos/data/

Etape 1 :

- Chargez les différents fichiers csv en utilisant la bibliothèques *pandas.*
- Affichez les informations qui représentent la taille des données de chaque fichier.
- Que font ces fonctionnalités de pandas ? (*head*, *description*, *info*).
- Affichez 5 échantillons de la variable contenant les données d'entraînement.

Etape 2 :

- Créez une fonction que vous appellerez *feature_extractor* qui appellera le model CNN *InceptionV3 pré-entrainé sur ImagetNet.*

Etape 3:

- Utilisez la fonction *StringLookup* de *Keras* afin de transformer les étiquettes (“tags”) de la variable contenant les données d'entraînement en entier.

Etape 4:

- Ajoutez à votre code la fonction *crop_center_square*.
  La fonction récupère la hauteur et la largeur de l’image. Par la suite, elle identifie la plus petite valeur entre les deux dimensions afin de découper l’image en un carré centré au milieu.

```
def crop_center_square(frame):
    y, x = frame.shape[0:2]
    min_dim = min(y, x)
    start_x = (x // 2) - (min_dim // 2)
    start_y = (y // 2) - (min_dim // 2)
    return frame[start_y : start_y + min_dim, start_x : start_x + min_dim]
```

Etape 5:

- Créez une fonction que vous nommerez *load_video*. Cette fonction aura 3 paramètres : le chemin de la vidéo, le nombre de frames, et la taille de redimensionnement de l’image (224,224).
  Vous utiliserez la fonction *cv2.VideoCapture* afin de charger la vidéo.
  A chaque frame de la vidéo, vous devez rogner votre image au centre en faisant appel à la fonction *crop_center_square* et faire un redimensionnement.
  Convertir la vidéo en un tableau *numpy*.

Etape 6:

- Ajoutez à votre code la fonction *process_video* qui aura comme paramètre le chemin vers une vidéo :
  Cette fonction fait appel à la fonction *load_video*.
  Elle initialise 2 matrices : *temp_frame_features* (pour stocker les caractéristiques des frames) et *temp_frame_mask* (pour indiquer quelles frames sont valides)
  Pour chaque frame (jusqu'à une longueur max), *process_video* fera appel à la fonction *feature_extractor* pour obtenir un vecteur de caractéristiques.
  Le masque sera mis à jour pour indiquer quelles frames ont été traitées.

```python
def process_video(args):
    path, root_dir = args
    paths = f"/kaggle/input/ucf101-videos/{os.path.join(root_dir, path)}"
    frames = load_video(paths)
    frames = frames[None, ...]

    temp_frame_mask = np.zeros(
        shape=(
            1,
            MAX_SEQ_LENGTH,
        ),
        dtype="bool",
    )
    temp_frame_features = np.zeros(
        shape=(1, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32"
    )

    for i, batch in enumerate(frames):
        video_length = batch.shape[0]
        length = min(MAX_SEQ_LENGTH, video_length)
        for j in range(length):
            temp_frame_features[i, j, :] = feature_extractor.predict(
                batch[None, j, :], verbose=0,
            )
        temp_frame_mask[i, :length] = 1  # Set the mask for valid frames

    return temp_frame_features.squeeze(), temp_frame_mask.squeeze()
```

Etape 7:

- Créez une fonction que vous nommerez *preapre_all_video* qui aura pour objectif de traiter plusieurs vidéos simultanément, en extrayant des caractéristiques pour chaque vidéo et en générant des masques, le tout parallélisé avec *ThreadPoolExecutor* :
  Cette fonction récupère les chemins des vidéos et les labels dans le DataFrame (df).
  Par la suite, elle utilisera la fonction *process_video* pour traiter chaque vidéo via un pool de threads (20 threads).

```python
with ThreadPoolExecutor(max_workers=20) as executor:
    results = list(tqdm(executor.map(process_video, [(path, root_dir) for path in video_paths]), total=len(video_paths), desc="Processing videos"))
```

tqdm est utilisé pour suivre l'avancement du traitement.
Les caractéristiques et masques doivent être stockés dans des arrays *numpy*, et les labels doivent être mis en forme pour être compatibles avec le modèle.

Etape 8 :

- Créez une fonction que vous nommerez *sequence_model*. Cette fonction aura deux entrées pour le modèle :

```
frame_features_input = keras.Input((MAX_SEQ_LENGTH, NUM_FEATURES))
mask_input = keras.Input((MAX_SEQ_LENGTH,), dtype="bool")
```

  Appliquez deux couches *TimeDistributed(Dense)* pour extraire des caractéristiques temporelles sur chaque frame. La première de 512 et la seconde de 256.
  Ajoutez trois couches *LSTM* avec des unités décroissantes (512, 256, 128).
  Ajoutez des couches *Dense* et *Dropout* pour éviter le surapprentissage, puis une couche de sortie *softmax* pour la classification.

  Utilisez également un *optimisateur* de type *Adam* avec un learning_rate de 0.0001

Etape 9:

- La fonction *run_experiment* que vous devrez créer aura comme paramètre les données d'entraînement et de validation ainsi que leurs labels.
  Faites appel à la fonction *keras.callbacks.ModelCheckpoint* afin que le modèle sauvegarde les poids lors de l'entraînement si la performance s'améliore.
  Cette fonction procédera également à la partie entraînement, vous prendrez d'ailleurs 30% des données pour la validation.
  Par la suite la fonction effectuera l'évaluation et affichera *l'accuracy*.
  Les prédictions du modèle sont converties en labels et un rapport de classification est généré avec *classification_report*.

Etape 10 :

- Importez LabelEncoder de sklearn afin de pouvertir les labels textuels train_labels et test_labels.

  pour cela vous aurez besoin de la fonction label_encoder.fit_transform

---

Etape 11:

- Lancez votre fonction *run_experiment*. Que pouvez-vous conclure par le rapport de classification ?

Etape 12:

- Affichez sur le même graphe la courbe des pertes d'entraînement et de validation
- Affichez sur le même graphe la courbe de précision pour l'entrainement et validation
- Que pouvez-vous conclure.

Etape 13 :

- Affichez la matrice de confusion.
- Que constatez-vous ?

Etape 14 :

- A partir des fonctions ci-dessous, faites un test sur une vidéo aléatoire se trouvant dans la variable contenant les vidéos tests.

```
def prepare_single_video(frames):
    frames = frames[None, ...]
    frame_mask = np.zeros(
        shape=(
            1,
            MAX_SEQ_LENGTH,
        ),
        dtype="bool",
    )
    frame_features = np.zeros(shape=(1, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32")

    for i, batch in enumerate(frames):
        video_length = batch.shape[0]
        length = min(MAX_SEQ_LENGTH, video_length)
        for j in range(length):
            frame_features[i, j, :] = feature_extractor.predict(batch[None, j, :])
        frame_mask[i, :length] = 1

    return frame_features, frame_mask
```

```python
def sequence_prediction(path):
    class_vocab = label_processor.get_vocabulary()
    tst = "test"
    paths = f"/kaggle/input/ucf101-videos/{os.path.join(tst, path)}"

    frames = load_video(paths)
    frame_features, frame_mask = prepare_single_video(frames)
    probabilities = sequence_model.predict([frame_features, frame_mask])[0]

    for i in np.argsort(probabilities)[::-1]:
        print(f"{class_vocab[i]}: {probabilities[i] * 100:5.2f}%")
    return frames
```

```python
def to_gif(images):
    converted_images = images.astype(np.uint8)
    imageio.mimsave("animation.gif", converted_images, duration=100)
    return Image("animation.gif")
```

- Que pouvez-vous conclure pour cet exemple ?
- Proposez des solutions afin d'améliorer les résultats pour la classification de la base de données UCF101.