Journal of Machine Learning Research 1 (2023) 0-0 Submitted 0/0; Published 0/0

Biomecánica: Análisis en tiempo real

Hugo C. Galván HCGALVAN@GMAIL.COM
Instituto AI
HUMAI
Buenos Aires, , ARGENTINA

Felipe Aguirre FELIPEAGUIRRE66@GMAIL.COM
Instituto AI
HUMAI
Buenos Aires, , ARGENTINA

Editor:

Abstract

.

Keywords: Biomecánica, MediaPipe, OpenCV, Machine Learning

1. Introducción

La biomecánica, evalúa si una postura corporal genera sobrecarga en la estructura del aparato locomotor humano. Las posturas correctas del cuerpo en movimiento como en reposo, están definidas en la higiene postural. La falta de higiene postural, llevan a problemas de salud y falta de bienestar general.

Las redes neuronales convulcionales de la visión, rama específica del aprendizaje profundo, emula el sistema visual humano. Esta arquitectura utiliza la visión artificial para identificar patrones, agrupando distintos modelos de aplicación. El grupo que investiga la estimación de la pose humana, ha logrado avances en rendimientos de eficiencia y bajo consumo para análisis en tiempo real.

Una de las tareas más difíciles en la visión por computadora es resolver el alto grado de configuración a realizar. Ya sea determinar la soltura del cuerpo humano con todas sus extremidades y la complejidad de resolver límites. Ya sea encontrar fronteras de partes similares o variaciones debido a la ropa, como el tipo de cuerpo, la iluminación, y muchos otros factores. Este problema se define como las técnicas de visión por computadora que predicen la ubicación de varios puntos clave humanos (articulaciones y puntos de referencia) como hombros, codos, muñecas, caderas, rodillas, tobillos y cuello.
En esta parte irán los artículos con estado de arte.

Sin embargo, diferentes estudios enfocaron su esfuerzo a generar arquitecturas optimizadas en reconocimiento de posturas y artículos específicos avanzaron en su aplicación biomecánica de los modelos con dos orientaciones. La primer orientación fue tratar con set de datos privados y arquitecturas rediseñadas y no disponibles; y la segunda orientación utilizaron la captura de videos e imágenes, para diseñar datos sintéticos con ecuaciones y algebra a partir de conocimientos previos en higiene postural. Por lo tanto, en este artículo,

propusimos diseñar un evaluador de posturas embebido en un app de escritorio, que integre los avances de optimización y las orientaciones particulares aplicado a la biomecánica deportiva. Específicamente utilizamos las tecnologías de MediaPipe Pose y Open Visión como red troncal para el procesamiento y captura de una entrada de video convencional y aplicamos la tecnología de Machine Learning para el modelado, justificado en la rápidez de cálculo y devolución en tiempo real.

2. Materiales y Métodos

2.1 MediaPipe

MediaPipe es una solución de google, para trabajar con vídeos e imágenes en tiempo real, con el fin de detectar objetos y elementos con contenido semántico (detección de la cara, skeleton tracking o estimación de la pose de objetos y personas, entre otros). se basa en una serie de grafos formados por una determinada cantidad de puntos de referencia o landmarks, dependiendo de la solución por la que se haya optado, unidos mediante un número de líneas. Actualmente MediaPipe cuenta con un total de dieciséis modelos basados en Machine Learning , así como la capacidad de implementarse en hasta seis lenguajes de programación. Para este proyecto se necesitará realizar un seguimiento y reconocimiento de una persona, por ende, la solución que se ha decidido utilizar ha sido la denominada Pose. El modelo MediaPipe Pose, utiliza una canalización Machine Learning *detector-tracker* de dos pasos. En el primer paso, el detector facilita el fotograma de entrada a la capa convolucional que produce un mapa de características convolucionales. Luego, una red neuronal utiliza este mapa para predecir la región. A partir de la región predicha se remodela, utilizando una capa de agrupación de región de interés (RoI). En el segundo paso, el tracker predice los 33 puntos claves contenidos en esta ROI. Para tener un rendimiento rápido, condensado en unos milisegundos por fotograma, y lograr en tiempo real la canalización de ML compuesto por la detección y seguimiento de la pose, se observa que la señal más fuerte a la red neuronal encima del torso es la cara de la persona (debido al alto contraste y pequeñas variaciones). En el caso de un video, el detector se ejecuta solo en el primer fotograma. Para los fotogramas siguientes, se deriva la ROI que contiene los puntos clave obtenidos anteriormente, como se describe en la figura siguiente

x e y: Coordenadas del punto de referencia normalizadas [0.0, 1.0] por el ancho y la altura de la imagen, respectivamente. z: Representa la profundidad del punto de referencia con la profundidad en el punto medio de las caderas como origen, y cuanto menor sea el valor, más cerca estará el punto de referencia de la cámara. La magnitud de z utiliza aproximadamente la misma escala que x. *visibility*: Un valor [0.0, 1.0] indica la probabilidad que el punto de referencia sea visible (presente y no presente) en la imagen. Pose se puede ejecutar en prácticamente cualquier dispositivo móvil, ordenador o incluso desde el propio navegador. Parámetros utilizados

Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5):
donde *min_detection_confidence* define valor mínimo de confianza ([0.0, 1.0]) del modelo de detección de personas para que la detección se considere exitosa. Predeterminado a 0.5.

min_tracking_confidence:
define valor mínimo de confianza dondel el mínimo es ([0.0, 1.0]), para que el modelo de

seguimiento de puntos de referencia se consideren rastreados correctamente, sino se invocará la detección en la siguiente imagen. Predeterminado a 0.5

La métrica de evaluación, utiliza el Porcentaje de Puntos Correctos con una tolerancia del 20%, donde asume que el punto se detecta correctamente si el error euclidiano 2D es menor que el 20% del tamaño del torso de la persona correspondiente.

Esta arquitecturaEl entrenamiento y la validación se realizaron en un conjunto de datos geodiversos de varias poses, muestreados de manera uniforme.

2.2 Computer Vision OpenCV

OpenCV es una solución escrita en C/C++, desarrollada por Intel, que esta dirigida fundamentalmente a la visión por computador en tiempo real. Entre sus bloques de funciones: a. estructuras y operaciones de matrices, grafos, árboles entre otros; b. procesamiento y análisis de imágenes; c. análisis de movimientos y seguimiento de objetos; d. adquisición de videos e interfaces gráficas de videos. Las operaciones básicas comienza con la captura de entrada de un video, sea un archivo o una cámara; define el número de filas o altura en pixeles, columnas o ancho en pixeles y canales; tipo de imagen, número de pixeles. este arreglo o matriz llamado frame o fotograma realiza alguna operación. Estas tareas pasan por detección de objetos, demarcar partes importantes, agregar texto o cualquier otro tipo de gráfica.

La métrica Intersection over Union (IoU) evalúa la corrección de una predicción. El valor varía de 0 a 1. Con la ayuda del valor de umbral de IoU, podemos decidir si una predicción es Verdadero Positivo, Falso Positivo o Falso Negativo que es la Matriz de Confusión. Mientras que en la segmentación de imágenes, se decide haciendo referencia a los píxeles de Ground Truth.

Parámetros utilizados con *cv2*objeto de OpenCv:
cvtColor(), convierte una imagen de un espacio de color, se lee como una matriz de *row (height) x column (width) x color (3)*, utilizando *COLOR_BGR2RGB* parte de los colores es BGR (azul, verde, rojo) y lo convierte a rojo, verde, azul. BGR y RGB no son espacios de color, convenciones para el orden de los diferentes canales de color, se apilan para formar un píxel. Mientras que BGR se usa consistentemente en OpenCV, la mayoría de las otras bibliotecas de procesamiento de imágenes usan el orden RGB por ello su necesaria conversion. *cap.read()*, devuelve dos valores, donde el primer valor es booleano, lo definimos como *ret*, indica si el marco se lee con éxito o no, y el segundo valor definido como *frame* es el fotograma real que se lee. *VideoCapture(0)*, captura la entrada de alguna cámara o archivo de video. *isOpened()*, verifica si está inicializado la cámara. *fourcc* es utilizado para comprimir los fotogramas, tiene 4 caracteres para definir el códec, ('P','I','M','1') es un códec MPEG-1; *fps* es para definir velocidad de fotogramas de la transmisión de video creada. *get(CAP_PROP_FPS)* o *get(CV_CAP_PROP_FPS)*: obtiene los fotogramas por segundo de la cámara. *get(cv2.CAP_PROP_FRAME_HEIGHT)*, *get(cv2.CAP_PROP_FRAME_WIDTH)*: obtiene valores de la altura y ancho del marco en píxeles de la cámara. *waitKey()*, espera un evento, en base a un tiempo en milisegundos, solo si hay una ventana activa. *release()*, libera el recurso, la cámara que está utilizando.

2.3 Train/test splitting

Es menester separar los datos en dos conjuntos: el de entrenamiento y el de testeo. Este ultimo debe utilizarse al FINAL del proyecto para garantizar una predicción no sesgada de la performance del modelo final.

La opción mas sencilla es utilizar *train_test_split*, donde especificamos el porcentaje de datos que separamos para testear.

2.4 Estandarización

Cuando tenemos muchas features continuas, podemos tener problemas de unidades. Para evitar eso y que el algoritmo no asigne importancias espurias, conviene estandarizar. Estandarizar es fijar una estrategia para pasar los valores al intervalo [0,1], [-1,1] o lo que sea. En particular, *StandardScaler* transforma a x en cantidad de desviaciones estandar de la media:

$x \to \frac{x-\mu}{\sigma}$

2.5 Evaluación del algoritmo

Estamos interesandos en problemas de clasificación en el que queremos aprender a asignar una clase $\mathcal{C}_k$ a un dato x teniendo como datos de entrenamiento un conjunto etiquetado x_{train}, t_{train}

Enfoque discriminativo.

El interés apunta a problemas de clasificación para aprender a asignar una clase $\mathcal{C}_k$ a un dato x teniendo como datos de entrenamiento un conjunto etiquetado x_{train}, t_{train}

El *Perceptron* es un clasificador que provee una *función discriminante* $y(x, w)$ que separa a las clases. Es decir, y tiene incluida una decisión sobre la clase a la que pertenece x, es decir modela priors.

En el enfoque *discriminativo*, el modelo busca aprender

$$p(\mathcal{C}_k|x)$$

En este enfoque no intenta modelar priors, verosimilitud y evidencia. Se centra directamente en el *posterior*. Ahora, la función $y(x, w)$ decide automáticamente la clase a la que pertenece x. Con el posterior a mano, requiere implementar la *teoría de la decisión* para asignar un valor de t. Al examinar la *Regresión Logística*, para dos clases, lleva a modelar el posterior como

$$p(\mathcal{C}_1|\vec{x}) = y(\vec{x}, \vec{w}) = \sigma(\vec{w}^T \vec{\phi}(\vec{x}))$$

La función sigmoide

Es necesario decidir la función de error adecuada para el problema y definir un algoritmo para minimizar dicha función.

Podemos encontrarla como hicimos para fundamentar cuadrados mínimos: escribiendo la verosimilitud.

t es una variable binaria, con valores exito o fracaso. Para una única medición, la verosimilitud $p(t|\vec{x},\vec{w})$ es del tipo Bernoulli $p(t|\mu) = \mu^t(1-\mu)^{1-t}$ con μ la probabilidad de éxito. ¿Cuál es dicha probabilidad de éxito? Justamente es lo que queremos obtener, $p(\mathcal{C}_1|\vec{x}) = y(\vec{x},\vec{w})$.

Entonces para un dataset $\vec{x}_n$, con $n = 1,..,N$ la verosimilitud es:
$p(T|X,\vec{w}) = \prod_{n=1}^{N} y_n^{t_n}(1-y_n)^{1-t_n}$, donde $y_n = y(x_n,\vec{w})$.

Tomando el logaritmo $Lnp(T|\vec{w}) = \sum_{n=1}^{N}(t_n Lny_n + (1-t_n)Ln(1-y_n))$. Aplicando un signo menos, llegamos a la función de error que nos interesa, y que vamos a usar muchisimo: la entropía cruzada o *cross-entropy*

$$E(\vec{w}) = -\sum_{n=1}^{N}(t_n lny_n + (1-t_n)ln(1-y_n))$$

Dado que $t_n = \{0,1\}$, la función de error hace lo siguiente:

Si $t_n = 1$, tiene en cuenta Lny_n. Entonces, y_n tiene que ser cercano a 1 para minimizar el error.

Si $t_n = 0$, tiene en cuenta $Ln1 - y_n$. Entonces, y_n tiene que ser cercano a 0 para minimizar el error.

Es decir, hace lo que tiene que hacer.

Recordemos que y_n no es t_n sino que es $p(t_n|\vec{x}_n,\vec{w})$.

Minimizar esta función de error es más difícil que minimizar la de cuadrados mínimos debido a la presencia de los logaritmos. Sin embargo, podemos definir un algoritmo iterativo para encontrar los $\vec{w}$, el Iterative Reweighted Least Squares o IRLS. Este algoritmo utiliza una actualización de Newton-Ralphson:

$\vec{w}^{nuevo} = \vec{w}^{viejo} - H^{-1}\nabla E(\vec{w})$

Donde $\nabla E(\vec{w})$ es el gradiente del error y H es la matriz Hessiana.

Para la regresión logística, uno puede llegar a sus propias ecuaciones normales iterativas aprovechando las propiedades de la sigmoide $\vec{w}^{nuevo} = (\Phi^T R\Phi)^{-1}\Phi^T Rz$

Con Φ la matriz de diseño, R la matriz diagonal cuyos elementos son $y_n(1-y_n)$ y z es un vector que se calcula como:

$z = \Phi\vec{w}^{viejo} - R^{-1}(Y-T)$

Con Y e T los vectores de predicciones y respuestas respectivamente. Noten que los pesos entran varias veces: en la matriz R, en el vector Y y explicitamente en z. Es por esto que es iterativo.

Este algoritmo también puede aplicarse al caso de regresión lineal, viendo que el algoritmo de Newton-Raphson converge a la solución cerrada en 1 paso. La diferencia acá es la función de activación sigmoide. Además, enfatizar que este algoritmo es iterativo pero no es secuencial, ya que utiliza todos los datos del dataset.

Asignando clases

La necesidad de decidir cuando asignar una medición a una clase, es importante.

Para el caso binario, podemos tomar el criterio que $x \in \mathcal{C}_1$ si

$$p(\mathcal{C}_1|x) \geq p(\mathcal{C}_2|x)$$

Dado que tenemos probabilidades que suman uno, este criterio pone la *frontera de decisión* en

$$p(\mathcal{C}_1|x) = 0.5$$

Métricas : Precisión/Exhaustividad

Puede mostrarse que esta elección de frontera de decisión maximiza la *exactitud* o *accuracy*

$$Exactitud = \frac{VP + VN}{VP + FP + VN + FN}$$

$$\text{accuracy} = \frac{Aciertos}{Totales}$$

Es decir, minimizamos la cantidad de *errores totales* (que no es lo mismo que minimizar la entropía cruzada).

Al ser un modelo discriminativo, los outliers o anomalias son menos problematicas.

Se puede ver el *accuracy de las predicciones positivas* o dicho de otro modo: *precision (precisión).*

$$\text{precision} = \frac{TP}{TP + FP}$$

Otra métrica importante Recall (Exhaustividad):

$$\text{recall} = \frac{TP}{TP + FN}$$

TP -¿ True positives (Verdaderos positivos): "*Era cinco y has dicho cinco*"

FP -¿ False positives (Falsos positivo): "*No era cinco y has dicho cinco*"

Si queremos más precisión genera un costo con menos exhaustividad, y viceversa.

Esto se llama *precision/recall trade-off* y tiene que ver con el umbral de decisión.

En el perceptrón para una instancia dada se calcula un valor (score), ese valor se pasa por la función de activación y si es menor que cero será de una clase y de la otra si es mayor. Si pudiésemos mover este umbral cambiaríamos la precisión y la exhaustividad.

Acknowledgments

Agradecer y reconocer el apoyo a este proyecto a profesores del Instituto HUMAI por los diferentes aportes y formación continua.

Appendix A.

En este apendice Section 6.2: