الخلاصة:

ساهم موضوع تحليل الإشارات البيولوجية عند الإنسان باستخدام تقنيات وخوارزميات التعلم العميق في حل الكثير من معضـلات التشـخيص وتلافي إعاقات بدنية كالبتر وحتى محاكاة الآلة لبعض الوظائف الحيوية، وتُعد إشـارة الدماغ EEG أعقد الإشـــارات الحيوية وأكثرها حملاً للميزات، ولدى علماء الحيوية الكثير من التســـاؤلات حول طبيعة النوم وتفاصـيل مراحله وعملياته وتأثيراته على الجسم وأعضـائه، وقد تم تقسـيم هذه المراحل إلى أربعة كل منها يصدر عنها إشارة مختلفة الميزات، وإن تحليل هذه الإشارات يُعد قفزة نوعية في الدراسات الحيوية لما له أثر على تشخيص عمل بعض الغدد الصماء بشكل خاص والتفاعل الجزيئي لباقي أعضاء الجسم بشكل عام، ولدى علماء التعلم العميق العديد من الخوارزميات المطبقة لتحليل هذه الإشـــارة مثل خوارزمية MFCC, HMM, CNN, LSTM والتي يصـــــدر عن كل منها نتائج معينة في التحليل تتبع لدقة النموذج وخرائط الميزات المسـتخرجة، وقد عمدنا إلى الجمع بين شـبكتي CNN, LSTM لدراسـة قاعدة بيانات معتمدة في أبحاث النوم تدعى EDFx تحوي إشـارات الدماغ المسـتخلصـة بواسـطة إلكترودات نقية، وكانت النتائج رفع دقة التصـنيف الشاملة مقارنة مع طرق أخرى للعمل، وتم العمل على بيئة Google Colab باستخدام لغة Python.

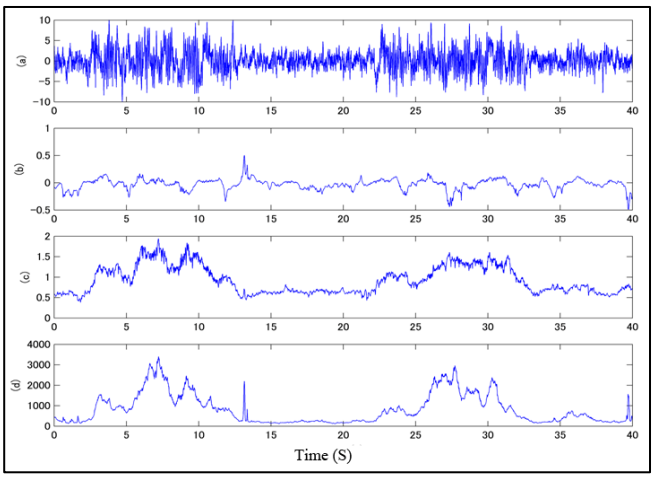
Abstract:

The topic of analysing human biological signals using deep learning techniques and algorithms has contributed to solving many diagnostic dilemmas and avoiding physical disabilities. Such as, amputation and even machine simulation of some biological functions. Its stages, processes and effects on the body and its organs. These stages have been divided into four, each of which emits a signal with different features, and the analysis of these signals is a quantum leap in biological studies because it has an impact on diagnosing the work of some endocrine glands in particular and the molecular interaction of the rest of the body's organs in general. Deep learning scientists have many algorithms applied to analyse this signal such as MFCC, HMM, CNN, LSTM algorithm, each of which produces certain results in the analysis that follow the model accuracy and extracted feature maps. We combined CNN and LSTM networks to study a sleep research database called EDFx that contains brain signals extracted by pure electrodes, and the results have increased the overall classification accuracy compared to other methods of work. The work is done by using Google Colab platform and the programming language was Python.

الكلمات المفتاحية: EEG analysis, Sleep stages classification, MFCC, HMM, CNN, LSTM.

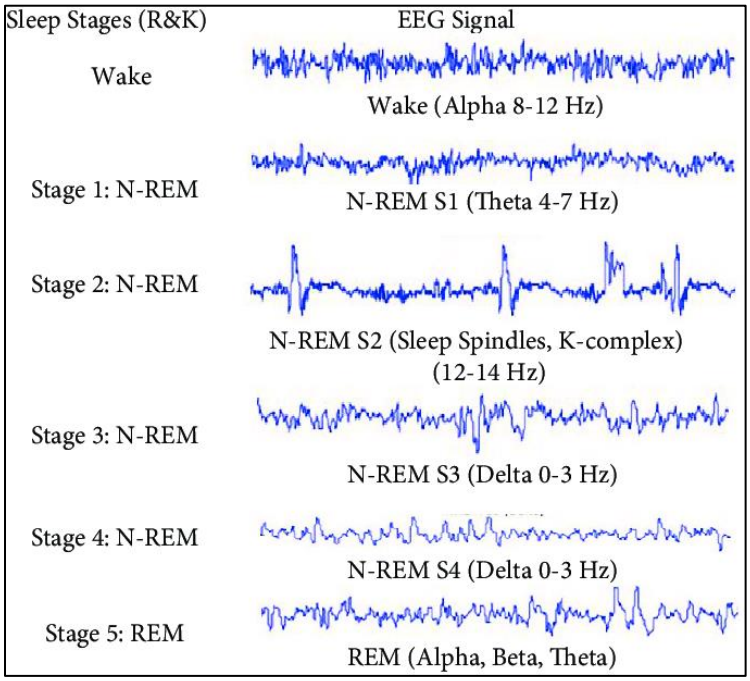
مقدمة:

تعتبر الإشارات من بنية الكون الدقيقة؛ إذا تعبر عن التغيرات الحاصلة في أجزائه ومكوناته الخاصة والتفاعلات مع الوسط المحيط أيضاً، ولهذه الإشارات أنواع عديدة كثيرة، منها الإشارات الحيوية في جسم الإنسان والتي يشرف على إدارتها الدماغ بشكل رئيسي وتنتج بشكل رئيسي من السيالة العصبية المتنقلة بين أعضاء الجسم والتغيرات الديناميكية في وبين الأعضاء، ومن أنواعها: الإشارة الكهربية الدماغية EEG (ElectroEncephaloGraphy)، والإشارة الكهربية القلبية ECG (ElectroCardioGraphy)، والإشارة الكهربية العضلية EMG (ElectroMyoGraphy)، والإشارة الكهربية العينية EOG (ElectroOphtamilyoGraphy)، وأيضاً الكثير، وقد نتج عن دراسة متغيرات حالات هذه الإشارات العديد من الدراسات والأبحاث المتعلقة بمحاكاة عمل الأعضاء المنتجة لها وتشخيص حالاتها، وإن دراسة إشارة الدماغ تعتبر من أعقد الدراسات لما لهذه الإشارة من تغيرات وحالات كثيرة جداً ومتداخلة بشكل كبير مع إشارات أخرى في الجسم وأبرزها إشارة القلب، ولعلنا ننوه إلى أن النبضة الواحدة من إشارة الدماغ تستهدف عدد من أعضاء الجسم المختلفة؛ إذ أنها تتحكم بالمستقبلات الحسية والحركية والترابطية والجسمية في القشرة الدماغية فقط، وتتحكم في الغدد الصماء والمخيخ والنخاع الشوكي في الداخل الفصي، وتتناقل بيانات الذاكرة وأوامر التفكير والمعالجة بين خلاياها، علاوة عن كون هذه الإشارة ذات كمون منخفض مقارنة مع إشارات أخرى كالراديو والأشعة، وإن من المعلوم بالضرورة أن تحليل أي إشارة ومعالجتها يؤدي إلى إخمادها بشكل كبير وإنهاكها حتى تفقد ميزاتها الرئيسية للعمل، عوضاً عن ضجيج أجهزة القياس لمستقبلات هذا النوع من الإشارات والذي يسبب بشكل كبير في تعقيد درجة التحليل وفصل إشارات الضجيج والتي تبرز بشكل أكبر من الإشارة المطلوب دراستها، ولهذا برز علم التعلم العميق في هذا المجال في تلافي مشكلة إجهاد هذه الإشارات من خلال استخراج خرائط الميزات المعنية بتعلم غايات عمل هذه الإشارات ووجهاتها بالمقارنة مع إشارات الأعضاء المعنية وموجات الاستقبال الخاصة بها، وتستخدم خوارزميات التعلم العميق لتحليل هذه الإشارات العديد من تقنيات التحسين والترشيح والتجميع والتحويلات الرياضية والتوزيعات الاحتمالية والتحليلات الإحصائية لتصنيف أنواعها المختلفة وتنقيتها من إشارات الضجيج والتشويش المطبق عليها كما يظهر في الشكل المجاور.



مشكلة البحث وأهدافه:

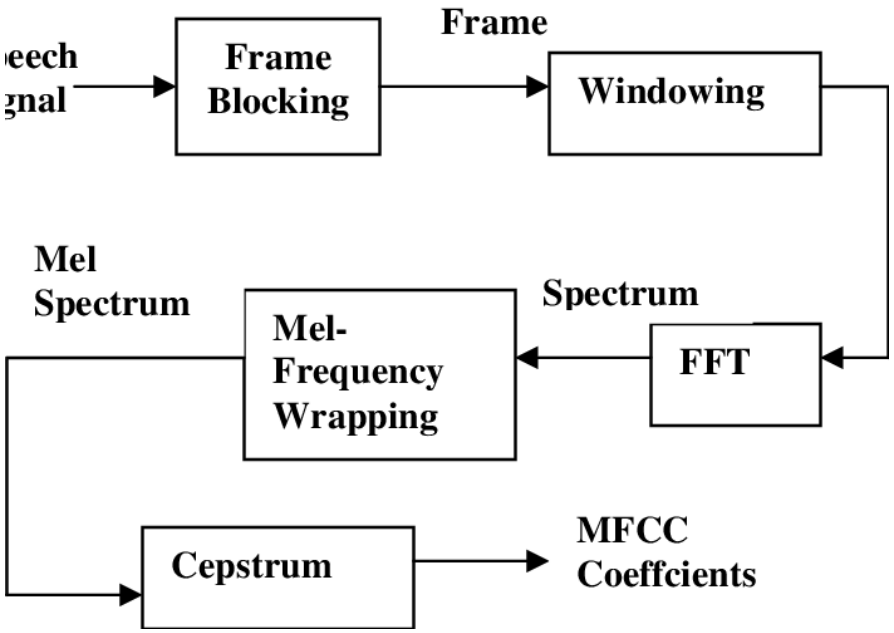
إن ضـــرر النوم الســـلبي أو قلته وعدم كفايته يصـــل حتى تأثر الدول بهذا النوع من الحرمان، لما له من تأثيرات اجتماعية واقتصادية على الأفراد بشكل خاص والمجتمع بشكل عام، وقد خلص الباحثان Gibson & Sharder إلى أن الموظفين الذين يقللون من نومهم يميل الوقت بســـاعة واحدة في المتوســـط إلى انخفاض رواتبهم بمقدار 4.9% على المدى الطويل، كما قدم الباحث Hafner وزملائه إلى أن التكاليف المجتمعية الناجمة عن الحرمان من النوم لها تأثير ســلبي بمقدار 2.28% للإنتاج الداخلي في الولايات المتحدة الأمريكية، والكثير من الدراســات التي توضح سلبية الموضوع ومخاطره على عدة أصعدة، وبالتالي لجأ العلماء إلى مراقبة عملية النوم والبحث عن المشاكل المتعلقة بها من خلال تصنيف فترات النوم إلى مراحل، وقد تم العمل في أحد الأبحاث يدعى AASM على تقسيم النوم إلى فترات من 30 إلى 60 ثانية تسمى العصور وتصنيفها إلى خمسة مراحل أساسية: مستيقظ (Awake) تكون فيه حركة العين غير ســـريعة (Non Rapid Eye Movement) Non-REM، والمرحلة الأولى من حركة العين الغير سريعة N1، والمرحلة الثانية N2، والمرحلة الثالثة N3، والمرحلة الرابعة هي مرحلة حركة العين السريعة REM، وتم ذلك البحث من خلال قياس إشارة EOG وأوامر التحكم بها من إشارة EEG، واســتخدمت العديد من الدراســات المتقدمة برعاية الحكومة الأمريكية تصــنيف هذه المراحل بتحليل قنوات كل من الإشــارات EEG, EOG, EMG وجمعها بقناة تحليل واحدة (والذي يعتبر أمراً بالغ الصــعوبة) تدعى Fpz-Cz والذي أعطى قدرة تحليلية ممتازة في بحث AASM، نهدف حالياً في العمل على تحليل قناة واحدة فقط لإشـــارة EEG بناء على قناة Fpz-Cz من البحث المذكور؛ وذلك من خلال الجمع بين مرحلتي تحليل لشـــبكتي CNN وLSTM المبنية على الشـــبكة العصـــبية المتكررة RNN، والغرض من هذا النظام هو تقديم نموذج مُصـــنف لإشـــارات EEG التي تعبر عن مراحل النوم المختلفة لضـمان فعالية ودقة أكبر للتعاون بين الآلة والإنســان في حل المشـــكلات، ويظهر الشـــكل المجاور هذه الإشارات والمراحل من موقع ResearchGate العالمي في بحث لتســجيل الإشــارة ودراســة منحنياتها بخوارزمية أقرب جوار K-NN.



خوارزمية MFCC:

هناك العديد من الخوارزميات المستخدمة في هذا النوع من الأعمال ولكل منها طريقته ودقته وتعقيده وسرعة تنفيذه الخاصـــة مثل خوارزمية تحليل المكونات الرئيســية PCA، وباســتعراض مبســط لخوارزمية معاملات تردد ميل الطيف المعكوس (Mel-Frequency Cepstrum Coefficients) MFCC فإن هذه الخوارزمية تعمل على الاستفادة من تقنيات أخرى وفق الخطوات التالية:

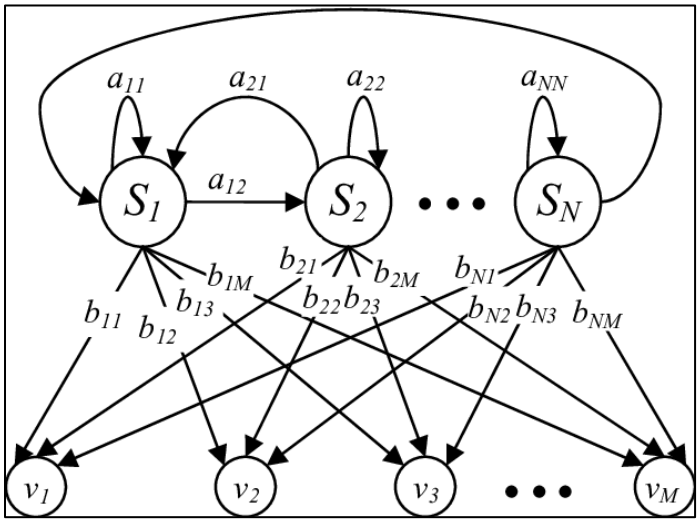
1. تقطيع الإشارة (Sampling) إلى عدة بلوكات (Blocks) للتعامل معها.
2. استخدام نافذة منزلقة (Sliding Window) للمرور على ميزات الإشارة المقطعة.
3. تطبيق تحويل فورييه (ويفضل تحويل فورييه السريع FFT).
4. عمل غلاف الإشـــارة يدعى غلاف الميل الترددي بتطبيق مرشـــح البنك (Bank Filter) الذي يســـتخدم مرشحات تمرير عالية ومنخفضة ومتوسطة (Low, Band, High Pass Filter) لكل طبقة ترشيح للإشارة.
5. استنتاج الطيف العكسي للإشارة من خلال معاملات الطيف المعكوس بتطبيق معادلة رياضية لتابع اللوغاريتم على تحويل فورييه السريع المستخدم والمتغيرات الناتجة عن الترشيح للطيف (Spectrum) واستنتاج الطيف المعكوس (Cepstrum)، وقد أتت تســمية الطيف المعكوس من طبيعة المعادلة التي تأخذ التابع العكســي لمتغيرات الترشــيح التي تســتخدم تحويل فورييه، ولغةً تم تســميته بعكس أول أربعة حروف من كلمة الطيف (Spectrum) لتصبح الطيف المعكوس (Cepstrum).



الشكل (1): مخطط البلوك لخوارزمية MFCC

خوارزمية HMMs:

تعتبر نماذج ماركوف المخفية (Hidden Markov Models) HMMs من الخوارزميات الهامة في علوم التعرف على الأنماط، فهي فئة من النماذج الرسومية الاحتمالية التي تسمح لنا بالتنبؤ بسلسلة من المتغيرات (المخفية) من مجموعة من المتغيرات المدروسة، وهي شبكة إحصائية بايزية Statical Bayesian Network غير متغيرة مع المزمن بذاتها وإنما بخطوات الخوارزمية نفسها وما ذلك إلا للتنبؤ بأفضل تسلسل للحالات المخفية، ويعود سبب تسمية حالات الدراسة بالمخفية هو أننا ندرس الحالة الحالية للنموذج بغض النظر عن تطوراته الماضية لنتنبأ بحالته المستقبلية الغير معروفة (المخفية)، ويوضح الشكل المجاور نموذج تسلسلي احتمالي للنماذج، بحيث نقوم بحساب احتمال وقوع عدة حالات في المرحلة الواحدة والحالات المنبثقة عنها بمرحلة أخرى بغض النظر عن الحالات التي تسبقها بتوزيع احتمالي مجاله [0,1] لكل حالة مرحلية ومن ثم نختار أفضل حالة للنموذج بناءً على ما هو مطلوب من خلال احتمالية الانتقال لحالة جديدة في المرحلة اللاحقة بشرط تحقق الحالة الحالية، واحتمالية الانتقال إلى حالة مرصودة مرتبطة بحالة مخفية، والاحتمال الأولي للانتقال إلى حالة مخفية، وتوضح المعادلات التالية المفاهيم السابقة رياضياً كما يلي:



The probability of observing a sequence

$$Y = y(0), y(1), \ldots, y(L-1)$$

of length L is given by

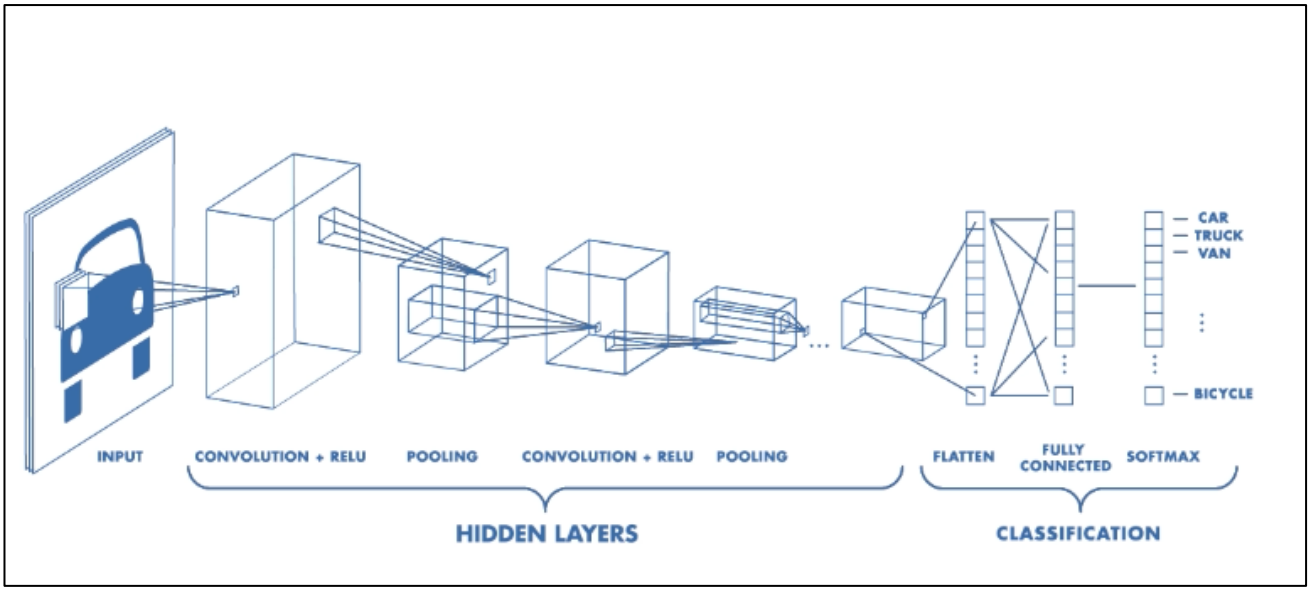
$$P(Y) = \sum_X P(Y \mid X) P(X),$$

where the sum runs over all possible hidden-node sequences

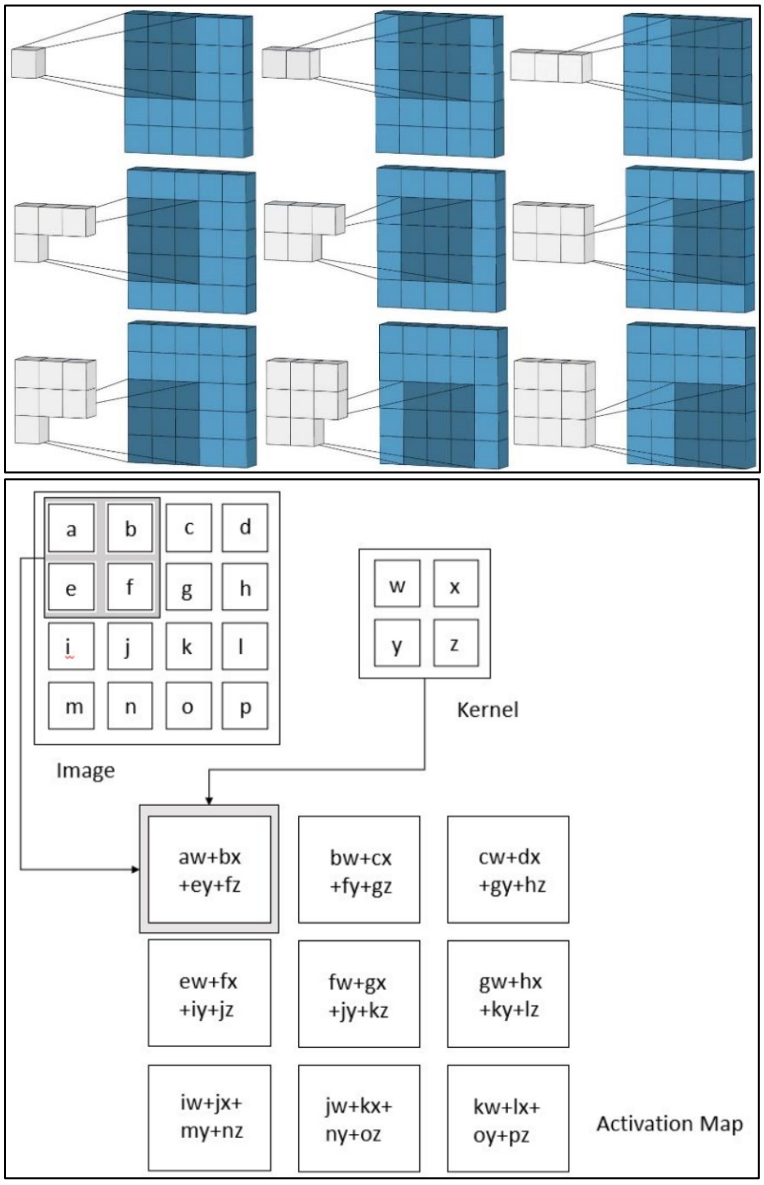
$$X = x(0), x(1), \ldots, x(L-1).$$

خوارزمية CNN:

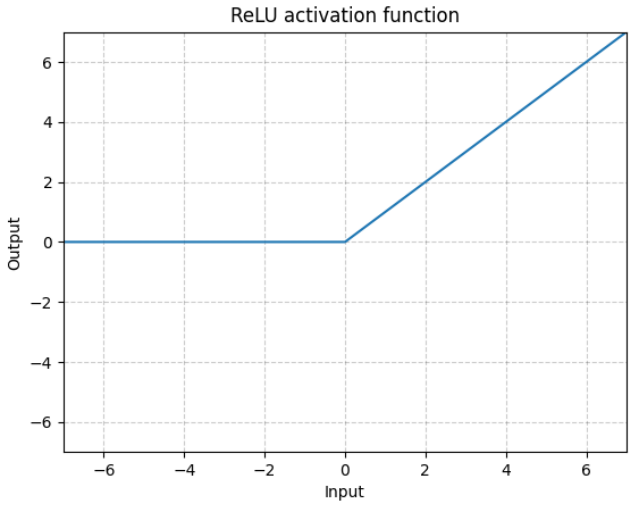
هي أحد أنواع الشبكات العصبية وسُميت بالشبكة العصبية التلافيفية أو الطي أو الانطواء (Convolutional) ورمزها ConvNet كونها تستخدم عملية جداء الانطواء الذي يشبه الجداء الديكارتي في عمله، وتكون فيها الطبقة المخفية مكونة من ثلاث طبقات: طبقة الطي، ودالة التفعيل (الترشيح)، وطبقة التجميع، وتكون طبقة الخرج هي عبارة عن طبقة التصنيف وتحتوي على طبقة التسطيح وطبقة الاتصال الكامل وطبقة التصنيف الأعلى كما يظهر الشكل التالي بنيتها المعمارية:



الشكل (2): يوضح البنية المعمارية للشبكة العصبية التلافيفية CNN

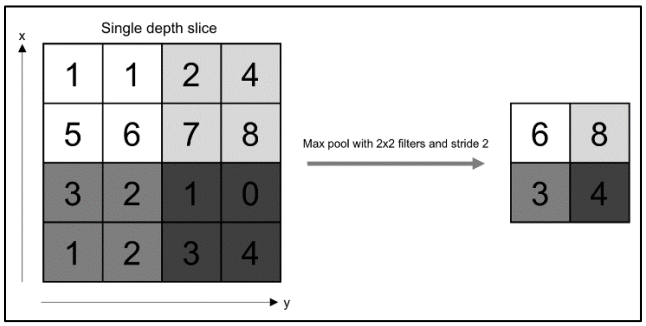


يوضح الشكل المجاور العملية التلافيفية في طبقات الشبكة، وهي اللبنة الأساسية والتي تحمل الجزء الرئيسي من الحمل الحسابي للشبكة، تعمل هذه الشبكة على إيجاد حاصل ضرب مصفوفتين ديكارتياً؛ حيث تمثل إحدى هذه المصفوفات الجزء المدروس من الصورة (كما نرى من الشكل ذو المربعات الزرق هي جزء من فضاء الدخل) والمصفوفة الأخرى تدعى مصفوفة النواة (Kernel) والتي تمثل مصفوفة الميزات، وتكون مصفوفة الدخل أكبر من مصفوفة النواة ولها عمق على الفضاء المدروس، ويسمى تتابع هذه العملية بالتمرير الأمامي، ويتم استخراج خريطة الميزات بضرب المصفوفة المُستنتجة سابقاً من العملية الانطوائية مع مصفوفة الدخل كما هو موضح في الشكل المجاور:

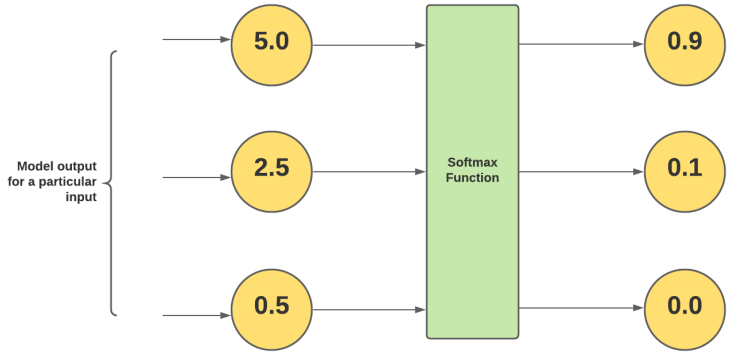


إن خرج العملية التلافيفية خطي يحتاج إلى تقويم غير خطي خاضع لعملية ترشيح غير خطية تدعى تابع التقويم الخطي ReLU (Rectified Linear Unit) وفي بعض الأحيان يدعى تابع التنشيط (Activation Function) ينتج عنه خريطة التنشيط المُرشحة، وهذا التابع يأخذ المنحني المجاور، وكما نرى فإنه يقوم بترشيح القيم الموجبة فقط من الخريطة وله عدة أنواع.

خرج الطبقة التلافيفية والترشيح يكون دخل لطبقة تدعى طبقة التجميع (Pooling Layer)، وتستخدم هذه الطبقة نوع من أنواع توابع التجميع ولعل أشهرها تابع التجميع للحد الأقصى (Max-Pooling Function) والذي يقوم بتقليص حجم خريطة الميزات أي تقليل المقدار المطلوب للحساب من خلال تقسيم مصفوفة الميزات إلى عدة مصفوفات جزئية وانتقاء أكبر قيمة لهذه المصفوفة (تحمل ميزات أكثر من الباقي عملياً) واستنتاج مصفوفة الميزات الجديدة كما يظهر الشكل المجاور.

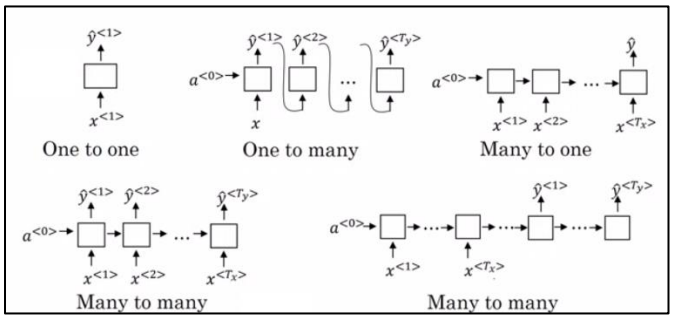


ويتم متابعة تكرار هذه العمليات حتى الخروج من الطبقة المخفية إلى طبقة الخرج التي تحتوي على طبقة التسطيح (Flatten Layer) والتي تقوم بتحويل كل المصفوفات ثنائية الأبعاد إلى مصفوفة أحادية البعد كمتجه خطي طويل، ومن ثم تدخل إلى طبقة الاتصال الكامل FCL (Fully Connected Layer) والتي تعمل على تجميع كل الميزات التي تم استنتاجها من خلال اتصال جميع خلايا الطبقة مع بعضها البعض، ومن ثم تدخل إلى طبقة التصنيف الأعلى (SoftMax Layer) والتي تعمل على تصنيف المدخلات بناءً على توزيع احتمالي واتخاذ القرار بالتصنيف بناءً على الاحتمال الأعلى من الخرج كما يظهر الشكل أدناه.



خوارزمية RNN:

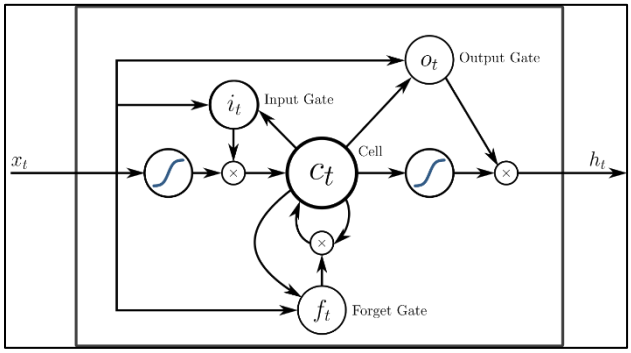
هي نوع من أنواع الشبكات العصبية التي تحتوي على ذاكرة داخلية ضمن الشبكة وتعتبر الأنسب للتطبيقات التي تستخدم البيانات المتسلسلة، ومعنى أنها تحتوي على ذاكرة داخلية أي أنها تقوم بتأدية الوظيفة نفسها لكل إدخال بيانات ولكن بالاعتماد على المخرجات السابقة في المراحل التي تسبق المرحلة المدروسة، وبهذا تتمكن من معالجة البيانات المتسلسلة بحيث ترتبط جميع المدخلات مع بعضها وهذا ما يجعل هذا النهج المستخدم الأنسب للتنبؤ بما هو تالياً، وبهذا يمكننا القول أن المرحلة المدروسة تقوم بمعالجة بيانات المرحلة الحالية بالإضافة إلى بيانات الماضي القريب في المرحلة السابقة، ولها عدة أنواع كما يظهر الشكل المجاور.

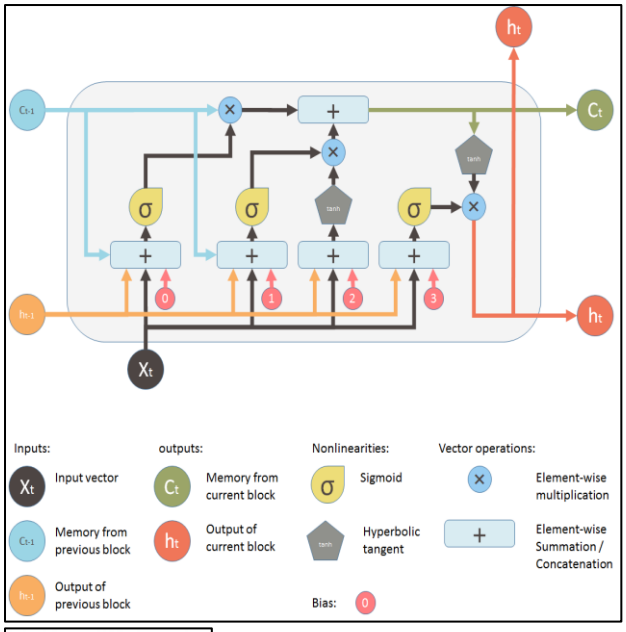


يعاني التعلم على الشبكات العصبية العميقة من مشكلة تدعى التدرج اللوني (Vanishing Gradients) وأخرى تدعى الانفجار اللوني (Exploding Gradients) عند القيام بعملية الانتشار الخلفي وإعادة ضبط الأوزان وهذا التدرج يقيس التغيير في جميع الأوزان (Weights) فيما يتعلق بالخطأ الذي يتم حسابه غالباً باستخدام تابع الخطأ (Loss Function)، فأثناء تدريب الشبكة المتكررة يمكن أن يصبح الخطأ صغيراً جداً أو كبيراً جداً ويكون الأداء سيئاً والدقة منخفضة والفترة التي استغرقتها الشبكة في التدريب طويلاً، ولتلافي هذه المشكلة نعمد إلى تهيئة الوزن أو اختيار وظيفة تنشيط صحيح أو اتباع نهج الذاكرة طويلة الأمد LSTM والتي تعتبر أفضل حل لمشكلة التدرج اللوني سواء كانفجار التدرج أو اختفائه.

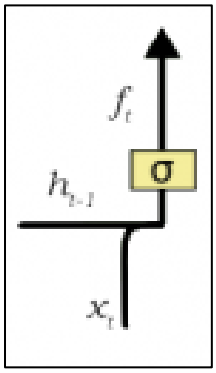
خوارزمية LSTM:

هي خوارزمية مشتقة من الشبكة العصبية المتكررة RNN وتُدعى خوارزمية أو نموذج الذاكرة طويلة المدى LSTM (Long Short Term Memory)، وعلاوةً على ما تم ذكره سابقاً فإن هذا النموذج يقوم بتذكر المعلومات الهامة في المدخلات عند مرحلة من مراحل المعالجة وذلك لأنها تحتوي على بوابة تحدد تدفق البيانات والذي يتيح لنا تعديل صغير في المعلومات عند إضافة معلومات جديدة وهذا غير موجود في الشبكة المتكررة الأصلية، بحيث تحدد البوابات البيانات المهمة التي من الممكن أن تكون مفيدة مرحلياً لاحقاً كما الشكل المجاور.

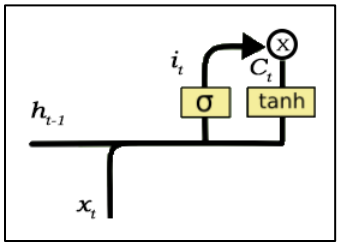




بوابة النســـيان (Forget Gate): تحدد هذه البوابة المعلومات الهامة والتي يجب تخزينها والمعلومات غير الهامة والتي يجب نسيانها مما يؤدي إلى تحسين الأداء بشكل كبير، ولدى هذه البوابة مدخلين أحدهما هو الخرج الناتج عن الخلية السابقة والآخر هو دخل الخلية الحالية، كما يتم إضافة ومضاعفة التحيز والأوزان المطلوبة ويتم تطبيق الدالة الســـينية (Sigmoid Function) على القيمة لأخذ القرار بشـــأن كونها مهمة للاحتفاظ بها عن القيمة 1، أم غير مهمة ونسيانها عند القيمة 0.



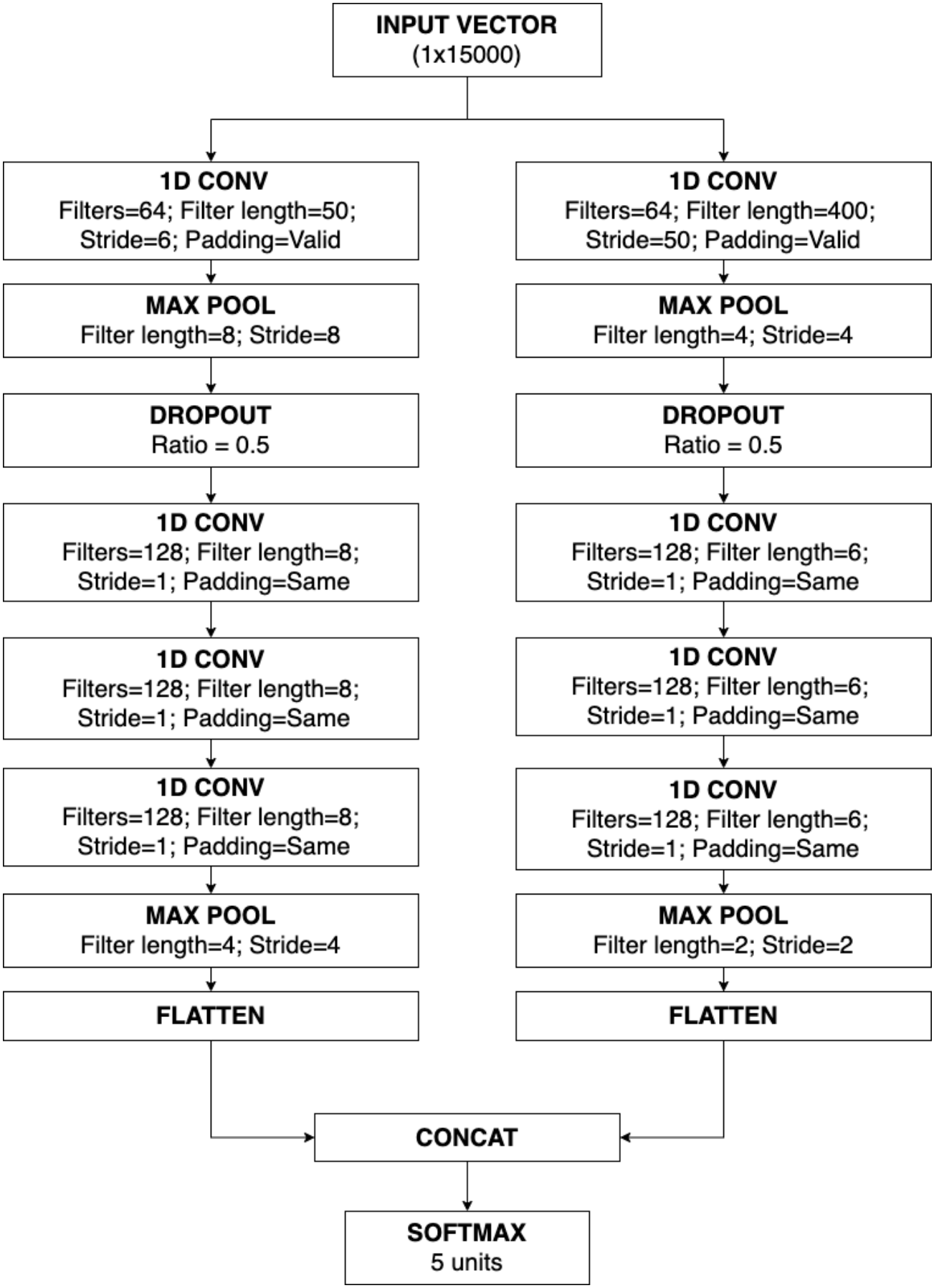
بوابة الدخل (Input Gate): تهتم بالدخل الخاص بالخلية وأيضـاً تسـتخدم تابع تنشـيط مثل التابع الســـيني، كما يقوم بإنشـــاء مجموعة من المعلومات التي يجب إضـــافتها ويتم ذلك باســـتخدام وظيفة تنشـــيط أخرى لتابع الظل المثلثي القطعي (Tanh) ذو المجال [-1,1]، ويظهر الشكل المجاور بوابة الدخل المذكورة.



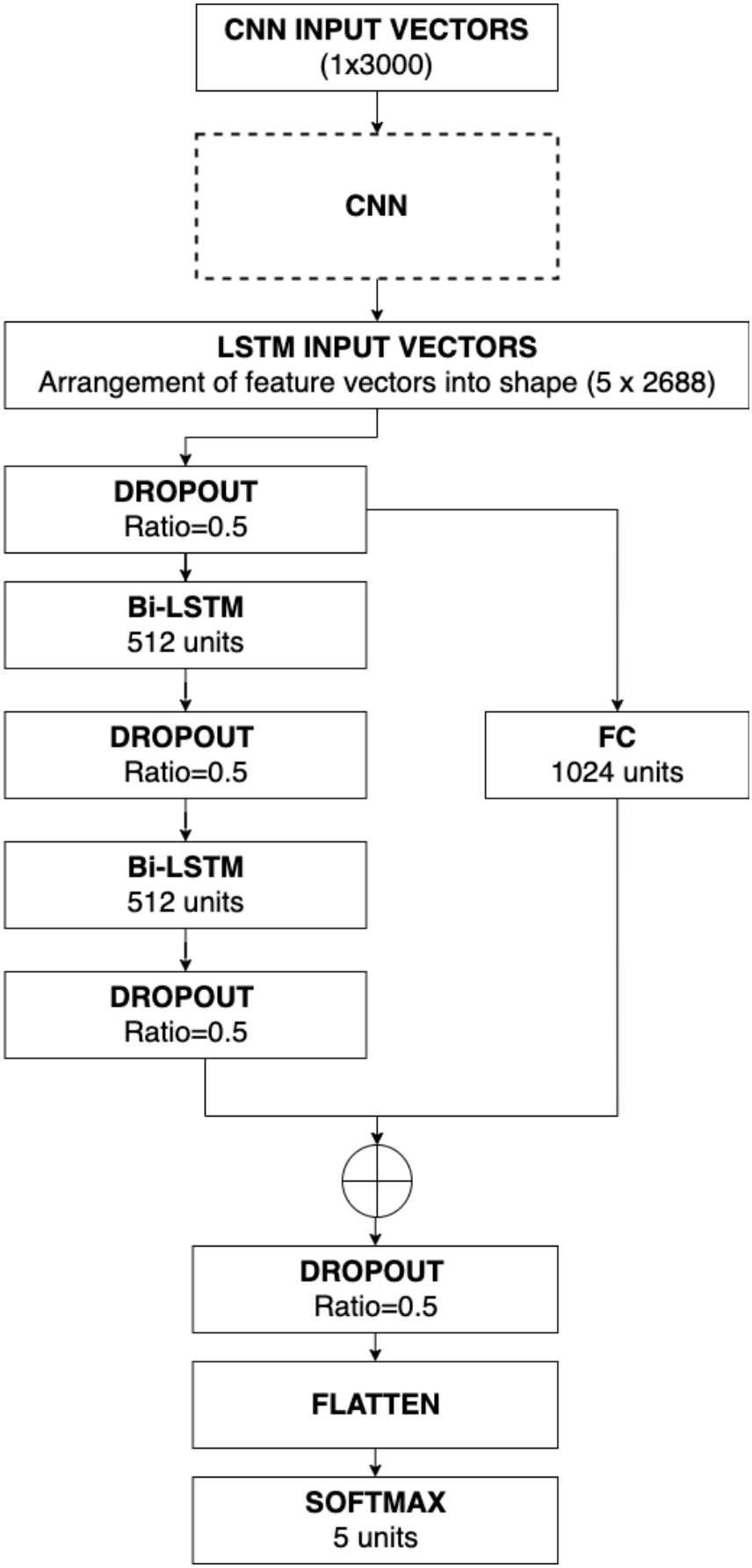
بوابة الخرج (Output Gate): هي البوابة المسؤولة عن اختيار المعلومات الهامة من الخلية الحالية وإظهارها كخرج، بحيث تنشـئ متجهاً للقيم باسـتخدام تابع الظل القطعي بالمجال القيمي له، وتســـتخدم التابع الســـيني لتجميع المدخلات الحالية والمخرجات السابقة لتقرير القيم التي يجب عرضها كما يظهر في الشكل المجاور.

نموذج CNN & LSTM:

تم الدمج بين شـبكتي التلافيف والتكرار بنموذج واحد، ولكن كانت التجربة الأولى أن نعمل على شـبكة التلافيف CNN على فرعين رئيسيين (أي الجمع بينهما Concatenation) أحدهما (الأيسر) يستخرج المعلومات الزمنية والتغيرات المفاجئة من سعة الإشارة بمرشحات وخطوات قصيرة، والآخر (الأيمن) يتعلم معلومات التردد من خلال العديد من المرشـحات والخطوات، وقد تم عمل تجربة باسـتخدام هذا النموذج فقط، ولكن كانت الأفضـلية للتجربة الثانية التي تســـتخدم بنية LSTM بالإضـــافة إلى النموذج الأول، وهذا ما يوضـــحه كلا المخططين تالياً مع ملاحظة تموضع طبقة التصنيف (SoftMax Layer) والتسطيح (Flatten Layer) لكلا النموذجين.

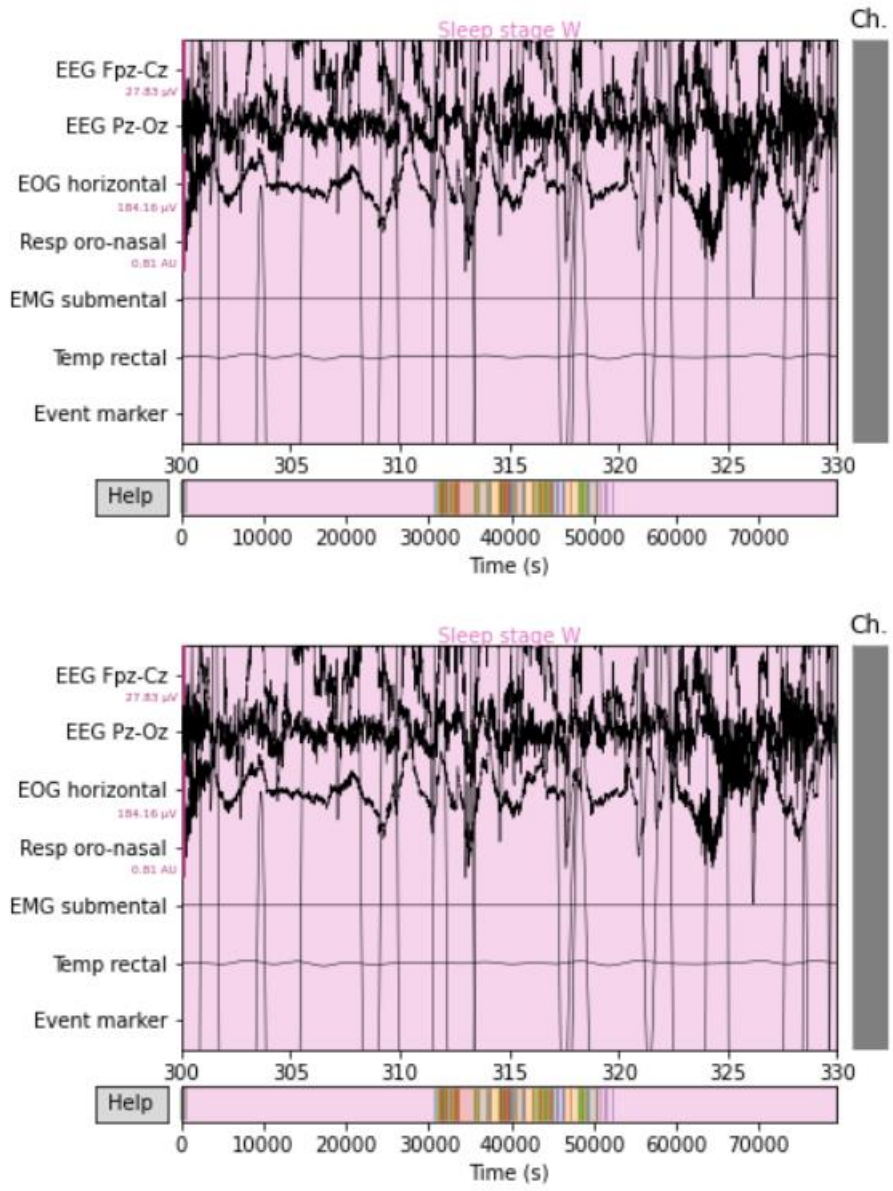


الشكل (3): يوضح بنية نموذج التجربة الأولى مع طبقة التصنيف الأخيرة SoftMax



الشكل (4): يوضح الدمج بين الشبكتين في نموذج واحد بعد إزالة طبقة التصنيف من نموذج CNN السابق ووضعها آخر طبقة في النموذج المطروح

النتائج:

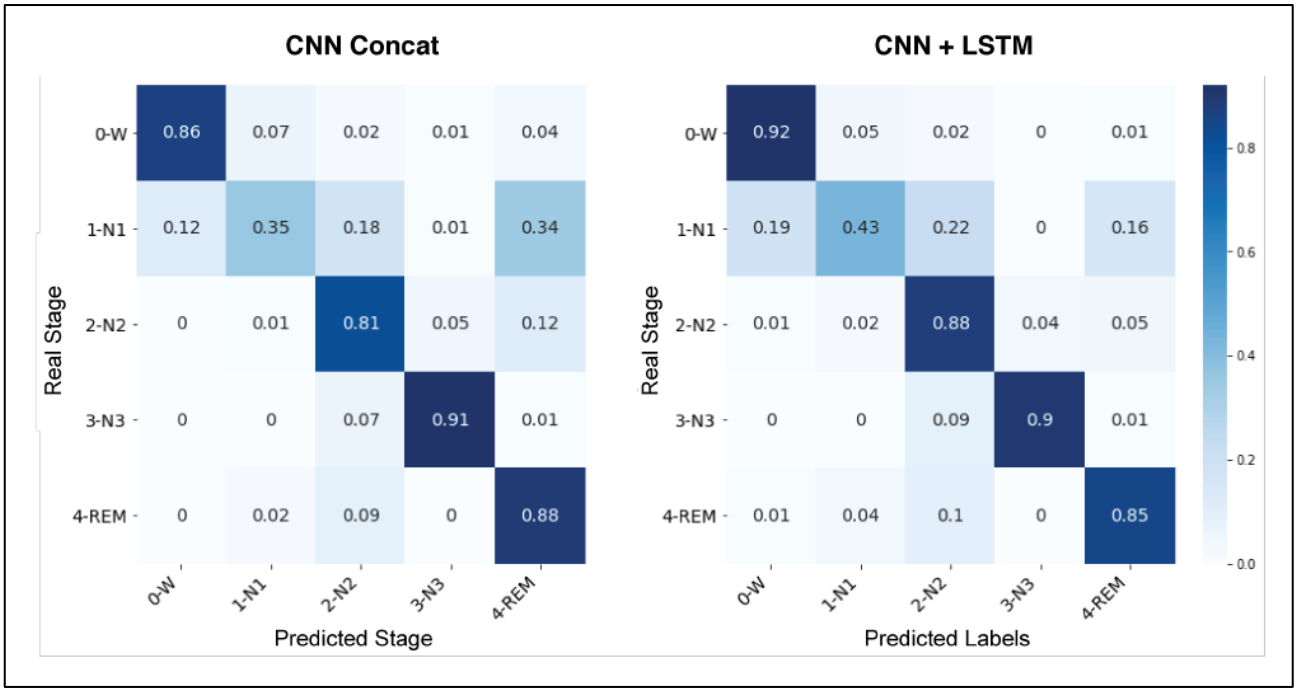


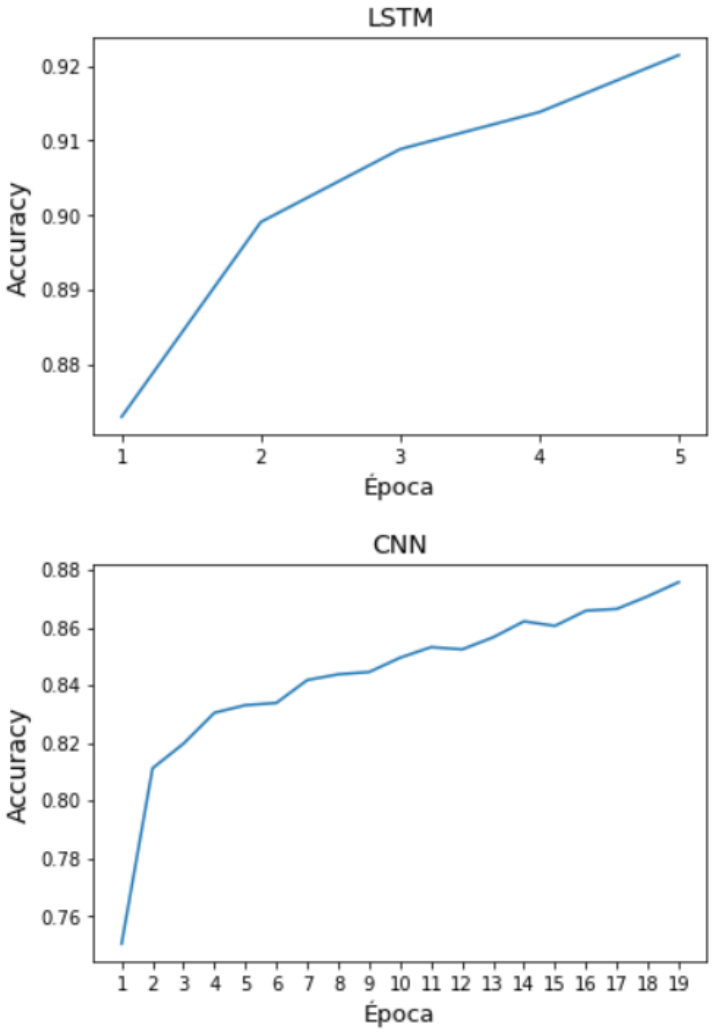
الشكل (5): Plotting the dataset

Sleep stage	Label	Quantity	Percentage of total
W	0	8,285	19.58
N1	1	2,804	6.63
N2	2	17,799	42.07
N3	3	5,703	13.48
REM	4	7,717	18.24
Total	-	**42,308**	**100**

تم تدريب النموذج على مجموعة بيانات Sleep-EDF التي تحتوي على 197 تســجيلاً للنوم طوال الليل وبحجم 8.1 جيجا بايت، وتم أخذ قناة Fpz-cz الخاصة بمنبع إشارة الدماغ EEG بتوزيع ملفات CSV على حالات النوم واليقظة كما يظهر الجدول المجاور.

وكانت مصفوفة ارتباك النموذج بالمقارنة مع الجمع بين شبكتي CNN وLSTM عوضاً عن الجمع بين شبكتي CNN متتاليتين (في التجربة الأولى) كما يلي:



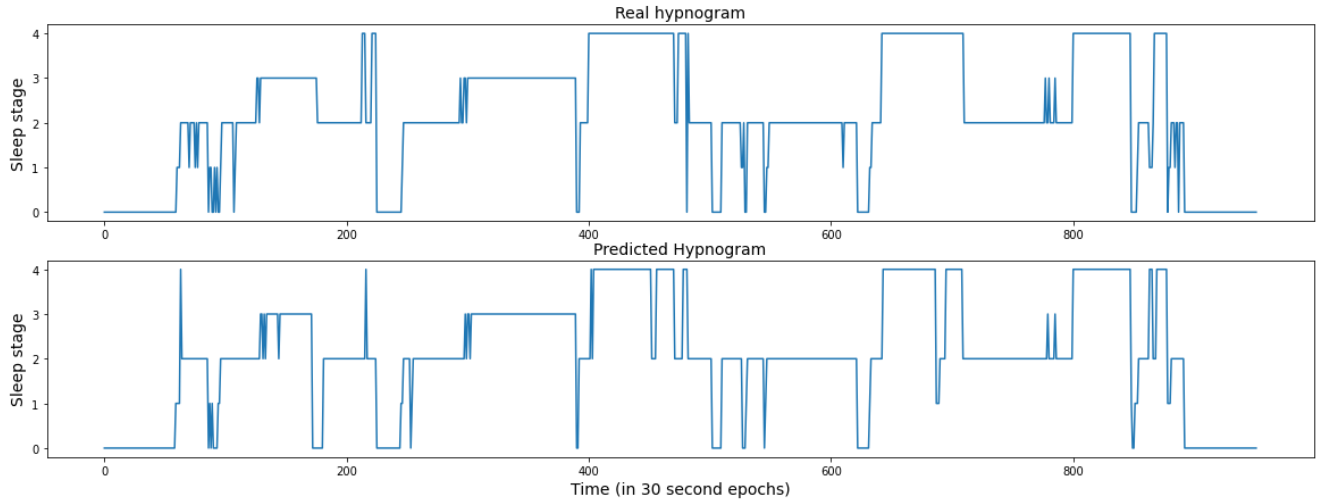


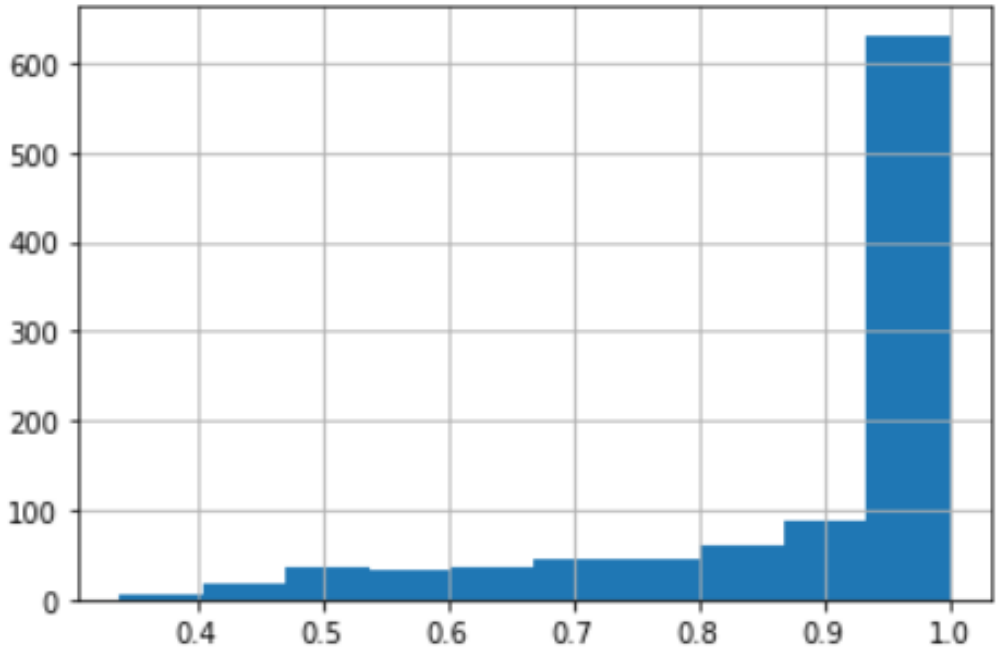
الشكل (6): يوضح منحنيات دقة النماذج لكل فترة تدريب

وبالمقارنة مع التشخيص البشري أظهر العمل نتائج فعّالة كما يوضح الجدول التالي:

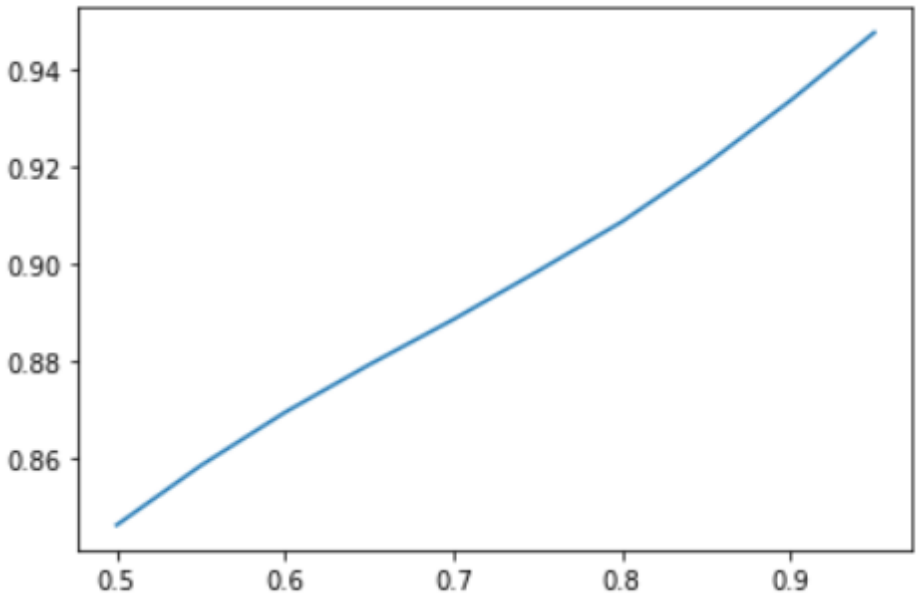
Experiment/ Reference	ACC	MF1	F1 per class					Training time (mm:ss)	Classification time
			0 - W	1 - N1	2 - N2	3 - N3	4 - REM		
CNN Concat	81.78	75.79	89.93	40.83	85.01	87.72	75.46	1:26	<1s
CNN + LSTM	85.30	79.56	91.40	46.77	88.02	88.78	82.82	2:18	<1s
Supratak *et al.* [17]	82.00	76.90	84.70	46.60	85.90	84.80	82.40	-	-
Human performance [12, 13]	80.00	-	-	-	-	-	-		2h - 4h

وكانت نتائج التصـنيف عندما اقترح النموذج عملية تنويم مغناطيسـي وفقاً لدراسـة حالات النوم فقد قام بتصـنيف المراحل بشكل ممتاز مقارنة مع النتائج التي أظهرها سابقاً دون الاقتراح، وتظهر النتائج كما يلي:





الشكل (7): احتمالات النموذج في فترات التصنيف



الشكل (8): نتائج تحليل العلاقة بين قيم العتبة (TV) ودقة نقاط البيانات لفترات النوم

	precision	recall	f1-score	support
0	0.9048	0.9690	0.9358	7000
1	0.5953	0.1522	0.2425	1005
2	0.9451	0.8865	0.9149	14453
3	0.9303	0.9423	0.9362	4729
4	0.8133	0.9459	0.8746	6659
accuracy			0.9013	33846
macro avg	0.8378	0.7792	0.7808	33846
weighted avg	0.8984	0.9013	0.8943	33846

الشكل (9): التقرير النهائي لنتائج التصنيف

الخاتمة:

أثبت هذا العمل أن المصـنف الآلي الذي يعتمد على شـبكتي CNN, LSTM قادر على تصـنيف مراحل النوم بدقة أعلى وسرعة أكبر من الإنسان، وإن زيادة عدد الطبقات التلافيفية في شبكة CNN من شأنه أن يعزز عملية التحليل واستخراج الميزات بشكل أكبر، وأنصح باستخدام نموذج DenseNet للطبقات التلافيفية الكثيفة أو نموذج ResNet لطبقات البلوكات المتبقية والذي من شأنهما أن يعززا أداء وسرعة هذه العملية على الوجه المطلوب.

References:

https://github.com/carlosfg97/AutomaticSleepStageClassifier

https://www.physionet.org/content/sleep-edfx/1.0.0/

https://jonathan-hui.medium.com/speech-recognition-feature-extraction-mfcc-plp-5455f5a69dd9

https://medium.com/@postsanjay/hidden-markov-models-simplified-c3f58728caab#:~:text=Hidden%20Markov%20Models%20(HMMs)%20are,that%20someone%20wears%20(observed)

https://medium.com/nybles/a-brief-guide-to-convolutional-neural-network-cnn-642f47e88ed4#:~:text=INTRODUCTION%3A,classification%2C%20object%20detection%2C%20etc

https://medium.com/analytics-vidhya/undestanding-recurrent-neural-network-rnn-and-long-short-term-memory-lstm-30bc1221e80d#:~:text=It%20is%20a%20type%20of,Siri%20and%20Googles%20Voice%20Search

الملحق:

Data Acquisition:

 # we install MNE library that will be of use for data manipulation.
!pip install mne

```
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime as dt
import os
import ntpath

import mne
from mne.datasets.sleep_physionet.age import fetch_data
from mne import Epochs, pick_types, find_events
from mne.io import concatenate_raws, read_raw_edf
```

#fetching data for subject 0, recording 1
[subject] = fetch_data(subjects=[0], recording=[1])

#Mapping of the channels

```
mapping = {'EOG horizontal': 'eog',
        'Resp oro-nasal': 'misc',
        'EMG submental': 'misc',
        'Temp rectal': 'misc',
        'Event marker': 'misc'}
```

 # creating raw_train object with the *PSG.edf file of fetched data.

```
raw_train = mne.io.read_raw_edf(subject[0])
```

#The annot_train object has the annotations (labels) to be extracted from *Hypnogram.edf file of fetched data.

```
annot_train = mne.read_annotations(subject[1])
raw_train.set_annotations(annot_train, emit_warning=False)
raw_train.set_channel_types(mapping) #mapeo de canales
```

 # plot some data

```
raw_train.plot(duration=30,scalings='auto', start=300 ) #start = 0, inicio de registro - W
```

#Each recording comes with metadata

```
print(raw_train.info)
```

#Sampling rate is an important variable

```
sampling_rate = raw_train.info['sfreq']

print('sampling_rate = {}'.format(sampling_rate))
```

#We define certain methods that will be useful for saving all the sleep recordings into files that will be later read as input data for the classifiers.

 # UTILS

 # This classes and methods have been extracted from Supratak et al. (2017) paper and code.

 # Please review https://github.com/akaraspt/deepsleepnet for further detail.

```
import re, datetime, operator
import numpy as np
from collections import namedtuple
from builtins import str

EVENT_CHANNEL = 'EDF Annotations'
```

```
class EDFEndOfData(BaseException): pass

def tal(tal_str):
  '''Return a list with (onset, duration, annotation) tuples for an EDF+ TAL stream.  '''

  exp = '(?P<onset>[+\-]\d+(?:\.\d*)?)' + \
    '(?:\x15(?P<duration>\d+(?:\.\d*)?))?' + \
    '(\x14(?P<annotation>[^\x00]*))?' + \
    '(?:\x14\x00)'

  def annotation_to_list(annotation):
    return annotation.split('\x14') if annotation else [] #str(annotation, 'latin_1')

  def parse(dic):
    return (
      float(dic['onset']),
      float(dic['duration']) if dic['duration'] else 0.,
      annotation_to_list(dic['annotation']))

  return [parse(m.groupdict()) for m in re.finditer(exp, tal_str)]


def edf_header(f):
  h = {}
  assert f.tell() == 0  # check file position
  assert f.read(8) == '0       '

  # recording info)
  h['local_subject_id'] = f.read(80).strip()
  h['local_recording_id'] = f.read(80).strip()

  # parse timestamp
  (day, month, year) = [int(x) for x in re.findall('(\d+)', f.read(8))]
  (hour, minute, sec)= [int(x) for x in re.findall('(\d+)', f.read(8))]
  h['date_time'] = str(datetime.datetime(year + 2000, month, day,
    hour, minute, sec))

  # misc
  header_nbytes = int(f.read(8))
  subtype = f.read(44)[:5]
  h['EDF+'] = subtype in ['EDF+C', 'EDF+D']
  h['contiguous'] = subtype != 'EDF+D'
  h['n_records'] = int(f.read(8))
```

```
  h['record_length'] = float(f.read(8))  # in seconds
  nchannels = h['n_channels'] = int(f.read(4))

  # read channel info
  channels = range(h['n_channels'])
  h['label'] = [f.read(16).strip() for n in channels]
  h['transducer_type'] = [f.read(80).strip() for n in channels]
  h['units'] = [f.read(8).strip() for n in channels]
  h['physical_min'] = np.asarray([float(f.read(8)) for n in channels])
  h['physical_max'] = np.asarray([float(f.read(8)) for n in channels])
  h['digital_min'] = np.asarray([float(f.read(8)) for n in channels])
  h['digital_max'] = np.asarray([float(f.read(8)) for n in channels])
  h['prefiltering'] = [f.read(80).strip() for n in channels]
  h['n_samples_per_record'] = [int(f.read(8)) for n in channels]
  f.read(32 * nchannels)  # reserved

  assert f.tell() == header_nbytes
  return h


class BaseEDFReader:
  def __init__(self, file):
    self.file = file


  def read_header(self):
    self.header = h = edf_header(self.file)

    # calculate ranges for rescaling
    self.dig_min = h['digital_min']
    self.phys_min = h['physical_min']
    phys_range = h['physical_max'] - h['physical_min']
    dig_range = h['digital_max'] - h['digital_min']
    assert np.all(phys_range > 0)
    assert np.all(dig_range > 0)
    self.gain = phys_range / dig_range


  def read_raw_record(self):
    '''Read a record with data and return a list containing arrays with raw
    bytes.
    '''
    result = []
```

```
  for nsamp in self.header['n_samples_per_record']:
    samples = self.file.read(nsamp * 2)
    if len(samples) != nsamp * 2:
      raise EDFEndOfData
    result.append(samples)
  return result


def convert_record(self, raw_record):
  '''Convert a raw record to a (time, signals, events) tuple based on
  information in the header.
  '''
  h = self.header
  dig_min, phys_min, gain = self.dig_min, self.phys_min, self.gain
  time = float('nan')
  signals = []
  events = []
  for (i, samples) in enumerate(raw_record):
    if h['label'][i] == EVENT_CHANNEL:
      ann = tal(samples)
      time = ann[0][0]
      events.extend(ann[1:])
      # print(i, samples)
      # exit()
    else:
      # 2-byte little-endian integers
      dig = np.fromstring(samples, '<i2').astype(np.float32)
      phys = (dig - dig_min[i]) * gain[i] + phys_min[i]
      signals.append(phys)

  return time, signals, events


def read_record(self):
  return self.convert_record(self.read_raw_record())


def records(self):
  '''
  Record generator.
  '''
  try:
    while True:
```

```
        yield self.read_record()
    except EDFEndOfData:
      pass


def load_edf(edffile):
  '''Load an EDF+ file.
  Very basic reader for EDF and EDF+ files. While BaseEDFReader does support
  exotic features like non-homogeneous sample rates and loading only parts of
  the stream, load_edf expects a single fixed sample rate for all channels and
  tries to load the whole file.
  Parameters
  ----------
  edffile : file-like object or string
  Returns
  -------
  Named tuple with the fields:
    X : NumPy array with shape p by n.
      Raw recording of n samples in p dimensions.
    sample_rate : float
      The sample rate of the recording. Note that mixed sample-rates are not
      supported.
    sens_lab : list of length p with strings
      The labels of the sensors used to record X.
    time : NumPy array with length n
      The time offset in the recording for each sample.
    annotations : a list with tuples
      EDF+ annotations are stored in (start, duration, description) tuples.
      start : float
        Indicates the start of the event in seconds.
      duration : float
        Indicates the duration of the event in seconds.
      description : list with strings
        Contains (multiple?) descriptions of the annotation event.
  '''
  if isinstance(edffile, basestring):
    with open(edffile, 'rb') as f:
      return load_edf(f)  # convert filename to file

  reader = BaseEDFReader(edffile)
  reader.read_header()

  h = reader.header
```

```
  # get sample rate info
  nsamp = np.unique(
    [n for (l, n) in zip(h['label'], h['n_samples_per_record'])
    if l != EVENT_CHANNEL])
  assert nsamp.size == 1, 'Multiple sample rates not supported!'
  sample_rate = float(nsamp[0]) / h['record_length']

  rectime, X, annotations = zip(*reader.records())
  X = np.hstack(X)
  annotations = reduce(operator.add, annotations)
  chan_lab = [lab for lab in reader.header['label'] if lab != EVENT_CHANNEL]

  # create timestamps
  if reader.header['contiguous']:
    time = np.arange(X.shape[1]) / sample_rate
  else:
    reclen = reader.header['record_length']
    within_rec_time = np.linspace(0, reclen, nsamp, endpoint=False)
    time = np.hstack([t + within_rec_time for t in rectime])

  tup = namedtuple('EDF', 'X sample_rate chan_lab time annotations')
  return tup(X, sample_rate, chan_lab, time, annotations)
```

#A for-loop iterates over all the available sleep recordings of the Sleep-EDF database and they are transformed into data structures that can later be the input for the classifiers.

#if you are using Google Colab, before writing any files, you need to mount your Google Drive to the notebook with this cell

```
from google.colab import drive
drive.mount('/content/gdrive')
```

#Mounted at /content/gdrive

 # replace next variable (input_data_path) with your own path

```
input_data_path = '/content/gdrive/My Drive/Automatic_sleep_classification/data'
```

 # we iterate over all recordings and subjects

```
for r in [1,2]:
```

```
  for i in [i for i in np.arange(20)]:

    if (i == 13) & (r ==2):
      continue


    [s1_files] = fetch_data(subjects=[i], recording=[r])
    select_ch = 'EEG Fpz-Cz'
    print('Files loaded: {} and {}'.format(s1_files[0], s1_files[1]))

    #We generate the object with signal data for each recording
    raw = read_raw_edf(s1_files[0], preload=True, stim_channel=None)
    sampling_rate = raw.info['sfreq']
    raw_ch_df  =  raw.to_data_frame(scaling_time=100.0)[select_ch]  #default="EEG
Fpz-Cz",
    raw_ch_df = raw_ch_df.to_frame()
    raw_ch_df.set_index(np.arange(len(raw_ch_df)))


    # we obtain the recording's metadata, or header.
    f = open(s1_files[0], 'r', encoding = 'latin_1')
    reader_raw = BaseEDFReader(f)
    reader_raw.read_header()
    h_raw = reader_raw.header
    f.close()
    raw_start_dt = dt.strptime(h_raw['date_time'], "%Y-%m-%d %H:%M:%S")

    f = open(s1_files[1], 'r')
    reader_ann = BaseEDFReader(f)
    reader_ann.read_header()
    h_ann = reader_ann.header

    _, _, ann = zip(*reader_ann.records())
    f.close()
    ann_start_dt = dt.strptime(h_ann['date_time'], "%Y-%m-%d %H:%M:%S")

    assert raw_start_dt == ann_start_dt

    # Label values
    W = 0
    N1 = 1
    N2 = 2
    N3 = 3
```

```
    REM = 4
    UNKNOWN = 5

    stage_dict = { "W": W, "N1": N1, "N2": N2, "N3": N3, "REM": REM, "UNKNOWN": UNKNOWN }
    class_dict = { 0: "W", 1: "N1",  2: "N2", 3: "N3", 4: "REM", 5: "UNKNOWN" }
    ann2label = {  "Sleep stage W": 0, "Sleep stage 1": 1,"Sleep stage 2": 2, "Sleep stage 3": 3,
                "Sleep stage 4": 3,  "Sleep stage R": 4, "Sleep stage ?": 5, "Movement time": 5 }
    EPOCH_SEC_SIZE = 30


    # Vector with indices and labels
    remove_idx = []    # indicies of the data that will be removed
    labels = []        # indicies of the data that have labels
    label_idx = []
    for a in ann[0]:
      onset_sec, duration_sec, ann_char = a
      ann_str = "".join(ann_char)
      label = ann2label[ann_str]
      if label != UNKNOWN:
        if duration_sec % EPOCH_SEC_SIZE != 0:
          raise Exception("Something wrong")
        duration_epoch = int(duration_sec / EPOCH_SEC_SIZE)
        label_epoch = np.ones(duration_epoch, dtype=np.int) * label
        labels.append(label_epoch)
        idx = int(onset_sec * sampling_rate) + np.arange(duration_sec * sampling_rate, dtype=np.int)
        label_idx.append(idx)

      else:
        idx = int(onset_sec * sampling_rate) + np.arange(duration_sec * sampling_rate, dtype=np.int)
        remove_idx.append(idx)

    labels = np.hstack(labels)


    if len(remove_idx) > 0:
      remove_idx = np.hstack(remove_idx)
      select_idx = np.setdiff1d(np.arange(len(raw_ch_df)), remove_idx)
    else:
```

```
  select_idx = np.arange(len(raw_ch_df))

# Only valid labels
label_idx = np.hstack(label_idx)
select_idx = np.intersect1d(select_idx, label_idx)


# If necessary, extra indices are removed
if len(label_idx) > len(select_idx):
    extra_idx = np.setdiff1d(label_idx, select_idx)
    # Vector tail is trimmed
    if np.all(extra_idx > select_idx[-1]):
        n_trims = len(select_idx) % int(EPOCH_SEC_SIZE * sampling_rate)
        n_label_trims = int(math.ceil(n_trims / (EPOCH_SEC_SIZE * sampling_rate)))
        select_idx = select_idx[:-n_trims]
        labels = labels[:-n_label_trims]


raw_ch = raw_ch_df.values[select_idx]

# Vector should be splittable into 30 second periods
if len(raw_ch) % (EPOCH_SEC_SIZE * sampling_rate) != 0:
    raise Exception("Something wrong")
n_epochs = len(raw_ch) / (EPOCH_SEC_SIZE * sampling_rate)

x = np.asarray(np.split(raw_ch, n_epochs)).astype(np.float32)
y = labels.astype(np.int32)

assert len(x) == len(y)

# We keep only section that correspond to sleep.
w_edge_mins = 30
nw_idx = np.where(y != stage_dict["W"])[0]
start_idx = nw_idx[0] - (w_edge_mins * 2) #60  epochs before sleep onset
end_idx = nw_idx[-1] + (w_edge_mins * 2) # 60 epochs after sleep is finished
if start_idx < 0: start_idx = 0
if end_idx >= len(y): end_idx = len(y) - 1
select2_idx = np.arange(start_idx, end_idx+1)
raw_datapoint = x.shape[0]
x = x[select2_idx]
selected_datapoints = x.shape[0]
y = y[select2_idx]
cal =  (raw_datapoint - selected_datapoints )*100/raw_datapoint
```

#print("Register went down from {} to {} or {} percent".format(raw_datapoint, selected_datapoints, np.round(cal,2)))

```
    filename = ntpath.basename(s1_files[0]).replace("-PSG.edf", ".npz")
    save_dict = {
        "x": x,
        "y": y,
        "fs": sampling_rate,
        "ch_label": select_ch,
        "header_raw": h_raw,
        "header_annotation": h_ann,
    }

    np.savez(os.path.join(input_data_path, filename), **save_dict)

    print('File {} was processed and saved'.format(s1_files[0]))

    print("\n========================================\n")
```

Classifiers:

#Libraries and functions that will be needed. Loading of the NPZ files generated in the first notebook.
 # Libraries

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import time
from datetime import datetime as dt
import pandas as pd
import os
import seaborn as sns

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import StratifiedKFold

from sklearn.preprocessing import Normalizer
```

#Functions

```
def print_confusion_matrix(confusion_matrix, class_names, figsize = (10,7), fontsize=14):

    df_cm = pd.DataFrame(
        confusion_matrix, index=class_names, columns=class_names,
    )
    fig = plt.figure(figsize=figsize)
    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(),  rotation=0,  ha='right', fontsize=fontsize)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right', fontsize=fontsize)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    return fig

def _load_npz_file(npz_file):
    """Load data and labels from a npz file."""
    with np.load(npz_file) as f:
        data = f["x"]
        labels = f["y"]
        sampling_rate = f["fs"]
    return data, labels, sampling_rate

def _load_npz_list_files(npz_files):
    """Load data and labels from list of npz files."""
    data = []
    labels = []
    fs = None
    for npz_f in npz_files:
        tmp_data, tmp_labels, sampling_rate = _load_npz_file(npz_f)
        if fs is None:
            fs = sampling_rate
        elif fs != sampling_rate:
            raise Exception("Found mismatch in sampling rate.")
        data.append(tmp_data)
```

```
        labels.append(tmp_labels)
    data = np.vstack(data)
    labels = np.hstack(labels)
    return data, labels
```

 # Also, mount google drive again to be able to read the files

```
from google.colab import drive
drive.mount('/content/gdrive')
```

 # Reading of Recordings
 # Load npz files

 # Change variable 'path' with own path

path = '/content/gdrive/My Drive/Automatic_sleep_classification/data' #the path where the npz files were saved in the first notebook.

#path = '/content/gdrive/My Drive/Tesis/Código/Data_nuevodp'

```
files = os.listdir(path)
filespath = []
for i in files:
  filespath.append(os.path.join(path,i))

X, Y = _load_npz_list_files(filespath)

X = X.reshape((X.shape[0], X.shape[1], 1))
print(X.shape, Y.shape)
```

#CNN Concatenation:

```
def cnn_builder(model_type):
  '''function that sets the CNN layers from input to keras.Model'''

  input_length = 15000 if model_type == 'CNN_CONCAT' else 3000

  inputs = keras.Input(shape=(input_length,1))
```

#left leg - short filters

```
  x = layers.Conv1D( filters=64, kernel_size=50, strides=6 ,
                    name = 'conv1')(inputs)
```

```
x = layers.BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.MaxPool1D(pool_size=8, strides=8, name='max1')(x)

x = layers.Dropout(rate=0.5  , name ='dropout1' )(x)

x = layers.Conv1D(filters=128, kernel_size=8, strides=1,
                  name = 'conv2', padding = 'same' )(x)
x = layers.BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv1D(filters=128, kernel_size=8, strides=1,
                  name = 'conv3', padding = 'same' )(x)
x = layers.BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv1D(filters=128, kernel_size=8, strides=1,
                  name = 'conv4', padding = 'same')(x)
x = layers.BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.MaxPool1D(pool_size=4, strides=4, name = 'max2')(x)
x = layers.Flatten()(x)

#right leg - longer filters
x2 = layers.Conv1D( filters=64, kernel_size=400, strides=50 ,
                  name = 'conv1_2')(inputs)
x2 = layers.BatchNormalization()(x2)
x2 = layers.ReLU()(x2)

x2 = layers.MaxPool1D(pool_size=4, strides=4, name='max1_2')(x2)

x2 = layers.Dropout(rate=0.5  , name ='dropout1_2' )(x2)

x2 = layers.Conv1D(filters=128, kernel_size=6, strides=1,
                  name = 'conv2_2', padding='same' )(x2)
x2 = layers.BatchNormalization()(x2)
x2 = layers.ReLU()(x2)

x2 = layers.Conv1D(filters=128, kernel_size=6, strides=1,
                  name = 'conv3_2', padding='same' )(x2)
x2 = layers.BatchNormalization()(x2)
```

```
  x2 = layers.ReLU()(x2)

  x2 = layers.Conv1D(filters=128, kernel_size=6, strides=1,
                name = 'conv4_2', padding='same' )(x2)
  x2 = layers.BatchNormalization()(x2)
  x2 = layers.ReLU()(x2)

  x2 = layers.MaxPool1D(pool_size=2, strides=2, name = 'max2_2')(x2)

  x2 = layers.Flatten()(x2)
  #concatenate both legs
  concat = layers.Concatenate(name = 'concat')([x, x2])

  #final layer as softmax
  outputs = layers.Dense(5, activation=tf.nn.softmax, name = 'soft') (concat)

  model = keras.Model(inputs=inputs, outputs=outputs, name='sleep')

  return model

n_split=5 # range between 10 - 20 folds

skf = StratifiedKFold(n_splits=n_split, random_state=47)

cv_results = []
k = 0
for train_index,test_index in skf.split(X, Y):
  begin_time_loop = dt.now()


  X_train,X_test=X[train_index],X[test_index]
  y_train,y_test=Y[train_index],Y[test_index]

  k = k+1
  a = np.zeros((3000, 1))

  X_train = np.insert(X_train, 0, a, axis=0)
  X_train = np.insert(X_train, 0, a, axis=0)
  X_train = np.insert(X_train, -1, a, axis=0)
  X_train = np.insert(X_train, -1, a, axis=0)
  X_test = np.insert(X_test, 0, a, axis=0)
  X_test = np.insert(X_test, 0, a, axis=0)
  X_test = np.insert(X_test, -1, a, axis=0)
```

```
  X_test = np.insert(X_test, -1, a, axis=0)

  X_train2 = []
  for i in range(len(X_train)-4):
    j = i + 2
    clip = np.array([X_train[j-2], X_train[j-1], X_train[j], X_train[j+1], X_train[j+2]]).reshape((15000,1))
    X_train2.append(clip)

  X_test2 = []
  for i in range(len(X_test)-4):
    j = i + 2
    clip = np.array([X_test[j-2], X_test[j-1], X_test[j], X_test[j+1], X_test[j+2]]).reshape((15000,1))
    X_test2.append(clip)

  X_train_2=np.array(X_train2)
  X_test_2=np.array(X_test2)

  model = cnn_builder()

  #compile and define Adam optimizer
  model.compile(loss=keras.losses.SparseCategoricalCrossentropy(),
            optimizer=keras.optimizers.Adam(learning_rate=0.001),
            metrics=['accuracy'])

  history = model.fit(X_train_2, y_train,
                  batch_size=100,
                  epochs=6, verbose= 0)


  y_prob = model.predict(X_test_2)

  y_pred = y_prob.argmax(axis=-1)

  k_acc = accuracy_score(y_test,y_pred)
  print(k_acc)

  elapsed = dt.now() - begin_time_loop
  print('Fold {} took {}'.format(str(k), str(elapsed)))

  fold_result = {'fold': k, 'time': elapsed,'y_test': y_test, 'y_pred': y_pred, 'y_prob': y_prob,
```

```
            'history': history.history, 'accuracy': k_acc }


  del inputs, outputs, x1, x2, concat, model # delete model objects to guarantee each fold is trained from zero

  cv_results.append(fold_result)
```

 # save results for further analysis when required

```
import pickle

path = '/content/gdrive/My Drive/Automatic_sleep_classification/results/cnnconcat.pkl'

with open(path, 'wb') as f:
    pickle.dump(cv_results, f)
```

 # unravel y_pred and y_test into unique vectors for classification report

```
y_pred=[]
y_test=[]
for i in cv_results:
  y_pred.extend(i['y_pred'])
  y_test.extend(i['y_test'])


print(classification_report(y_test,y_pred, digits=4) )
```

 # print confusion matrix

```
C = confusion_matrix( y_test ,y_pred)
cn = np.transpose( np.transpose(C) / C.astype(np.float).sum(axis=1) )
cn = np.round(cn, 2)

df_cm = pd.DataFrame(
    cn,    index=['0-W','1-N1','2-N2','3-N3','4-REM'],    columns=['0-W','1-N1','2-N2','3-N3','4-REM'])
fig = plt.figure(figsize=(8,7))
try:
    heatmap = sns.heatmap(df_cm, annot=True, cmap = "Blues", annot_kws={"size": 14})
except ValueError:
    raise ValueError("Confusion matrix values must be integers.")
```

```
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right', fontsize=14)
plt.ylabel('True labels', fontsize = 14)
plt.xlabel('Predicted labels', fontsize=14)
```

#CNN & LSTM

```
def lstm_builder():

  inputs = keras.Input(shape=(5,2688))

  x = layers.Dropout(rate=0.5  , name ='lstm_dropout1' )(inputs)

  x1 = Bidirectional(layers.LSTM(512, return_sequences=True)) (x)
  x1 = layers.Dropout(rate=0.5  , name ='lstm_dropout2' )(x1)

  x1 = Bidirectional(layers.LSTM(512)) (x1)
  x1 = layers.Dropout(rate=0.5  , name ='lstm_dropout3' )(x1)

  x2 = layers.Dense(1024)(x)
  x2 = layers.BatchNormalization()(x2)
  x2 = layers.ReLU(name = 'lstm_residual1')(x2)

  addition = layers.Add(name = 'lstm_concat1')([x1, x2])


  x3 = layers.BatchNormalization()(addition)
  x3 = layers.ReLU(name = 'lstm_residual2')(x3)

  x3 = layers.Dropout(rate=0.5  , name ='lstm_dropout4' )(addition)

  x3 = layers.Flatten()(x3)

  outputs = layers.Dense(5, activation=tf.nn.softmax, name = 'lstm_soft') (x3)

  model_lstm = keras.Model(inputs=inputs, outputs=outputs, name='lstmsleep')

  return model_lstm
```

```
from tensorflow.keras.layers import Bidirectional #we will be needing this layer
```

 # perform training and prediction for 20 stratified folds

```
skf = StratifiedKFold(n_splits=5) #10 - 20

cv_results = []
k = 0
for train_index,test_index in skf.split(X, Y):
  begin_time_loop = dt.now()
  print('Current time: {}'.format(str(begin_time_loop)))

  k = k+1
  X_train,X_test=X[train_index],X[test_index]
  y_train,y_test=Y[train_index],Y[test_index]


  #CNN
  model = cnn_builder( model_type = 'CNN+LSTM')

  model.compile(loss=keras.losses.SparseCategoricalCrossentropy(),
            optimizer=keras.optimizers.Adam(learning_rate=0.001),
            metrics=['accuracy'])

  history1 = model.fit(X_train, y_train,
                batch_size=100,
                epochs=19, verbose=0) # 24


  #get feature vectors
  layer_name = 'concat'
  intermediate_layer_model = keras.Model(inputs=model.input,
                           outputs=model.get_layer(layer_name).output)
  intermediate_output_train = intermediate_layer_model.predict(X_train)
  intermediate_output_test = intermediate_layer_model.predict(X_test)
  X_train_lstm = intermediate_output_train
  X_test_lstm = intermediate_output_test

  a = np.zeros((1, 2688))

  X_train_lstm = np.insert(X_train_lstm, 0, a, axis=0)
  X_train_lstm = np.insert(X_train_lstm, 0, a, axis=0)
  X_train_lstm = np.insert(X_train_lstm, -1, a, axis=0)
```

```
X_train_lstm = np.insert(X_train_lstm, -1, a, axis=0)

X_test_lstm = np.insert(X_test_lstm, 0, a, axis=0)
X_test_lstm = np.insert(X_test_lstm, 0, a, axis=0)
X_test_lstm = np.insert(X_test_lstm, -1, a, axis=0)
X_test_lstm = np.insert(X_test_lstm, -1, a, axis=0)

X_train_lstm2 = []
for i in range(len(X_train_lstm)-4):
  j = i + 2
  clip = np.array([X_train_lstm[j-2], X_train_lstm[j-1], X_train_lstm[j],
X_train_lstm[j+1], X_train_lstm[j+2]])
  X_train_lstm2.append(clip)

X_test_lstm2 = []
for i in range(len(X_test_lstm)-4):
  j = i + 2
  clip = np.array([X_test_lstm[j-2], X_test_lstm[j-1], X_test_lstm[j],
X_test_lstm[j+1], X_test_lstm[j+2]])
  X_test_lstm2.append(clip)

X_train_lstm3=np.array(X_train_lstm2)
X_test_lstm3=np.array(X_test_lstm2)

#LSTM
lstm = lstm_builder()

lstm.compile(loss=keras.losses.SparseCategoricalCrossentropy(),
        optimizer=keras.optimizers.Adam(learning_rate=0.001),
        metrics=['accuracy'])

history2 = lstm.fit(X_train_lstm3, y_train,
              batch_size=100,
              epochs=5, verbose=0) #

test_scores = lstm.evaluate(X_test_lstm3, y_test, verbose=0)
print('Test accuracy:', test_scores[1])

y_prob = lstm.predict(X_test_lstm3)

y_pred = y_prob.argmax(axis=-1)

elapsed = dt.now() - begin_time_loop
```

```
  print('Fold {} took {}'.format(str(k), str(elapsed)))

  fold_result = {'fold': k, 'time': elapsed,'y_test': y_test, 'y_pred': y_pred, 'y_prob': y_prob,
                 'history_cnn': history1.history, 'history_lstm': history2.history,  'accuracy': test_scores[1] }


  cv_results.append(fold_result)

import pickle

path = '/content/gdrive/My Drive/Automatic_sleep_classification/results/cnnlstm.pkl'

with open(path, 'wb') as f:
    pickle.dump(cv_results, f)
```

 # unravel y_pred and y_test into unique vectors for classification report

```
y_pred=[]
y_test=[]
for i in cv_results:
  y_pred.extend(i['y_pred'])
  y_test.extend(i['y_test'])


print(classification_report(y_test,y_pred, digits=4) )
```

accuracy is higher when trained around 20 folds

 # print confusion matrix

```
C = confusion_matrix( y_test ,y_pred)
cn = np.transpose( np.transpose(C) / C.astype(np.float).sum(axis=1) )
cn = np.round(cn, 2)

df_cm = pd.DataFrame(
    cn,  index=['0-W','1-N1','2-N2','3-N3','4-REM'],  columns=['0-W','1-N1','2-N2','3-N3','4-REM'])
fig = plt.figure(figsize=(8,7))
try:
    heatmap = sns.heatmap(df_cm, annot=True, cmap = "Blues", annot_kws={"size": 14})
except ValueError:
```

```
    raise ValueError("Confusion matrix values must be integers.")
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right', fontsize=14)
plt.ylabel('True labels', fontsize = 14)
plt.xlabel('Predicted labels',  fontsize=14)
```

#average time of training+prediction per fold

```
elapsed_t = []
for i in cv_results:
  elapsed_t.append(i['time'])

elapsed_t = pd.Series(elapsed_t)
elapsed_t.mean()
```

#if data neews to be uploaded again, this cell helps

```
import pickle
with.open('/content/gdrive/My Drive/Automatic_sleep_classification/results/cnnlstm.pkl', 'rb') as f:
    data = pickle.load(f)
```

 # plot accuracy curves per training epoch. Uncomment lines if want figures to be saved

```
accuracies = []
accuracies.extend(data[3]['history_cnn']['accuracy'])
accuracies.extend(data[3]['history_lstm']['accuracy'])

plt.plot(data[3]['history_lstm']['accuracy'])
plt.title('LSTM', fontsize =14)
plt.ylabel('Accuracy', fontsize=14)
plt.xlabel('Época' , fontsize=13)
plt.xticks(ticks = np.arange(0, 5, step=1),  labels =  np.arange(1, 6, step=1))
#plt.savefig( 'fill-with-path' ,bbox_inches='tight')
plt.show()

plt.plot(data[3]['history_cnn']['accuracy'])
plt.title('CNN', fontsize =14)
plt.ylabel('Accuracy', fontsize=14)
plt.xlabel('Época' , fontsize=13)
plt.xticks(ticks = np.arange(0, 19, step=1),  labels =  np.arange(1, 20, step=1))
```

```
#plt.savefig( 'fill-with-path' ,bbox_inches='tight')
plt.show()

import random
import datetime
path = '/content/gdrive/My Drive/Automatic_sleep_classification/data'
files = os.listdir(path)
filespath = [] #lista de archivos npz
for i in files:
  filespath.append(os.path.join(path,i))


def one_record_test_set(filespath,i):

  files = filespath[:]
  test_record = files.pop(i)
  print('Using record {} as test set'.format(test_record))

  X_test, y_test = _load_npz_list_files([test_record])
  X_train, y_train = _load_npz_list_files(files)

  print(X_test.shape, y_test.shape, X_train.shape, y_train.shape)

  return X_train, X_test, y_train, y_test, test_record

cv_results = []

for i in range(1):
  i =  -7 #we select this recording because it produces an accuracy score comparable to the general accuracy when trained with cross val
  X_train, X_test, y_train, y_test, record = one_record_test_set(filespath, i)

  begin_time_loop = datetime.datetime.now()
  print('Current time: {}'.format(str(begin_time_loop)))

    #CNN

  model = cnn_builder(model_type = 'CNN+LSTM')

  model.compile(loss=keras.losses.SparseCategoricalCrossentropy(),
            optimizer=keras.optimizers.Adam(learning_rate=0.001),
            metrics=['accuracy'])
```

```
history1 = model.fit(X_train, y_train,
                batch_size=100,
                epochs=19, verbose=0) # 24


#get feature vectors
layer_name = 'concat'
intermediate_layer_model = keras.Model(inputs=model.input,
                                outputs=model.get_layer(layer_name).output)
intermediate_output_train = intermediate_layer_model.predict(X_train)
intermediate_output_test = intermediate_layer_model.predict(X_test)
X_train_lstm = intermediate_output_train
X_test_lstm = intermediate_output_test


a = np.zeros((1, 2688))

X_train_lstm = np.insert(X_train_lstm, 0, a, axis=0)
X_train_lstm = np.insert(X_train_lstm, 0, a, axis=0)
X_train_lstm = np.insert(X_train_lstm, -1, a, axis=0)
X_train_lstm = np.insert(X_train_lstm, -1, a, axis=0)

X_test_lstm = np.insert(X_test_lstm, 0, a, axis=0)
X_test_lstm = np.insert(X_test_lstm, 0, a, axis=0)
X_test_lstm = np.insert(X_test_lstm, -1, a, axis=0)
X_test_lstm = np.insert(X_test_lstm, -1, a, axis=0)

X_train_lstm2 = []
for i in range(len(X_train_lstm)-4):
  j = i + 2
  clip = np.array([X_train_lstm[j-2], X_train_lstm[j-1], X_train_lstm[j],
X_train_lstm[j+1], X_train_lstm[j+2]])
  X_train_lstm2.append(clip)

X_test_lstm2 = []
for i in range(len(X_test_lstm)-4):
  j = i + 2
  clip = np.array([X_test_lstm[j-2], X_test_lstm[j-1], X_test_lstm[j],
X_test_lstm[j+1], X_test_lstm[j+2]])
  X_test_lstm2.append(clip)

X_train_lstm3=np.array(X_train_lstm2)
X_test_lstm3=np.array(X_test_lstm2)
```

```
  #LSTM
  lstm = lstm_builder()

  lstm.compile(loss=keras.losses.SparseCategoricalCrossentropy(),
            optimizer=keras.optimizers.Adam(learning_rate=0.001),
            metrics=['accuracy'])

  history2 = lstm.fit(X_train_lstm3, y_train,
                    batch_size=100,
                    epochs=5, verbose=0) #

  test_scores = lstm.evaluate(X_test_lstm3, y_test, verbose=0)
  print('Test accuracy:', test_scores[1])

  y_prob = lstm.predict(X_test_lstm3)

  y_pred = y_prob.argmax(axis=-1)

  elapsed = datetime.datetime.now() - begin_time_loop
  print('Fold {} took {}'.format(str(1), str(elapsed)))

  fold_result = {'record': record, 'time': elapsed,'y_test': y_test, 'y_pred': y_pred,
'accuracy': test_scores[1] }


  cv_results.append(fold_result)

 # 0.8341862559318542

y_pred = cv_results[0]['y_pred']
y_test = cv_results[0]['y_test']

from seaborn import lineplot

fig, axs = plt.subplots(2,1, figsize=(20,7))

axs[0].set_title('Real hypnogram', size = 14)
axs[0].set_ylabel('Sleep stage', fontsize=14)
axs[0].set_yticks([0,1,2,3,4])


axs[1].set_title('Predicted Hypnogram', size = 14)
```

```
axs[1].set_ylabel('Sleep stage', fontsize=14)
axs[1].set_yticks([0,1,2,3,4])

axs[1].set_xlabel('Time (in 30 second epochs)', fontsize=14)

sns.lineplot(np.arange(len(y_test)), y_test, ax = axs[0])
sns.lineplot(np.arange(len(y_pred)), y_pred, ax = axs[1])
```

 # fig.savefig('fill-with-path',bbox_inches='tight')

Threshold Analysis:

#Libs and data loads

```
import pickle
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
import pandas as pd
import seaborn as sns
import numpy as np

from google.colab import drive
drive.mount('/content/gdrive')

path = '/content/gdrive/My Drive/Automatic_sleep_classification/results/cnnconcat.pkl'
```
#fill with own path

```
with open(path, 'rb') as f:
    data = pickle.load(f)

y_scores = []
y_preds = []
y_trues = []
for i in data:
  y_scores.extend(i['y_prob'])
  y_preds.extend(i['y_pred'])
  y_trues.extend(i['y_test'])
```

 # First, lets see the probabilities produces by the classifier

```
scores = pd.Series(y_scores)
scores = scores.apply(lambda x: max(x))
scores.sample(n=1000).hist()
```

#The proposal is that these weaker predictions should receive an in-depth analysis from a human scorer.

```
len(scores)
```

 # Prepare the dataframe

```
df = pd.DataFrame({'predicted':y_preds, 'true':y_trues, 'probs':scores })

import seaborn as sns
from sklearn.metrics import accuracy_score

%matplotlib inline
```

 # We analyze the relationship between threshold values (TV) and accuracy of the remaining data points (sleep epochs)

```
acc_scores = []
ths = [0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95]
for i in ths:
  dff = df[df.probs > i].reset_index()
  acc = accuracy_score(dff.true, dff.predicted)
  #print(acc)
  acc_scores.append(acc)

sns.lineplot(ths, acc_scores)

o_size = df.shape[0]

th = np.percentile(df.probs,20)  # change the percentile parameter to set percentage of epochs to filter out (5, 10, 20)
print('The threshold value is: ' + str(th)[0:5]+ '\n')
dff = df[df.probs > th].reset_index()
left = df[df.probs <= th].reset_index()

new_size = dff.shape[0]

print( str((o_size - new_size)) + ' sleep epochs were filtered out or discarded (sent to manual review). \n')

print('The resulting classification report is: \n \n', classification_report(dff.true, dff.predicted, digits=4))
```