Clasificación de rostros a partir de máquinas de vectores de soporte y redes neuronales

Classification of faces from support vector machines and neural networks

Autor 1: David Cediel Gómez Autor 2: Juan Pablo Narvaez, Autor 3: Santiago Londoño
Facultad de Ingenierías, Universidad Tecnológica de Pereira, Pereira, Colombia
Correo-e: david.cediel@utp.edu.co juan.narvaez@utp.edu.co santiago.londono@utp.edu.co

***Resumen*— En el campo del Machine Learning las redes neuronales y el aprendizaje profundo se han popularizado de una manera increíble, y no es de sorprenderse, ya que el desempeño de estas es muy elevado. Sin embargo, otro tipo de modelos como las máquinas de vectores de soporte o modelos Bayesianos se han visto opacados. En este artículo se resaltan las características de dos modelos de redes neuronales: Perceptrón multicapa y Red neuronal convolucional, también se explorará un modelo llamado máquinas de vectores de soporte y se mencionan las ventajas que tiene este modelo con respecto a las redes neuronales.**

***Palabras clave*— aprendizaje, optimización, kernel, vector, capa, neurona, convexa, gamma, características, cuadrícula, búsqueda, parámetros, coeficiente, umbral.**

***Abstract*— In the field of Machine Learning, neural networks and deep learning have become popular in an incredible way, and it is not surprising, since their performance is very high. However, other types of models such as support vector machines or Bayesian models have been opaque. This article will highlight the characteristics of two models of neural networks: Multilayer Perceptron and convolutional Neural Network, a model called support vector machines will also be explored and the advantages of this model with respect to neural networks will be mentioned.**

***Key Word* — learning, optimization, kernel, vector, layer, neuron, convex, gamma, characteristics, grid, search, parameters, coefficient, threshold.**

I. INTRODUCCIÓN

Las redes neuronales son ampliamente usadas en la industria, una de estas aplicaciones es el procesamiento de imágenes, para ello es necesario un tipo de red neuronal llamada red neuronal convolucional, esta será explicada de manera detallada a lo largo del artículo, se explicará qué es, qué es una capa de convolución y a que se refiere el término de Max Pooling.

Fecha de Recepción: 25-Nov-2019
Fecha de Aceptación:

Otra forma de hacer un procesamiento de imágenes es realizar un análisis de componentes principales (PCA por sus siglas en inglés), este término también será explicado detalladamente.

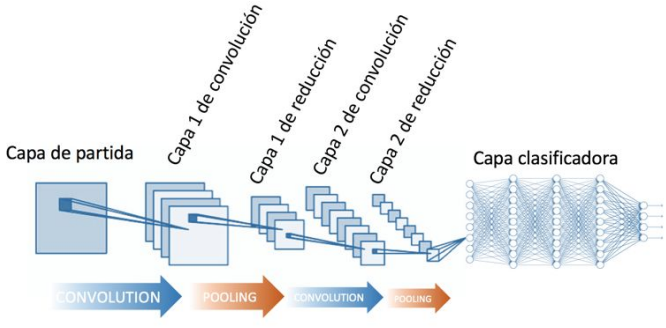
Como se mencionó anteriormente, en el Machine Learning existen modelos diferentes a las redes neuronales, en este artículo se explicará a detalle el funcionamiento de una máquina de vectores de soporte.

Por último, se mostrarán los resultados obtenidos que también están condensados en el código fuente del proyecto.

II. CONTENIDO

A. REDES NEURONALES CONVOLUCIONALES

Las redes neuronales convolucionales son un tipo de red neuronal que se basan en las neuronas la corteza visual primaria de un cerebro real.



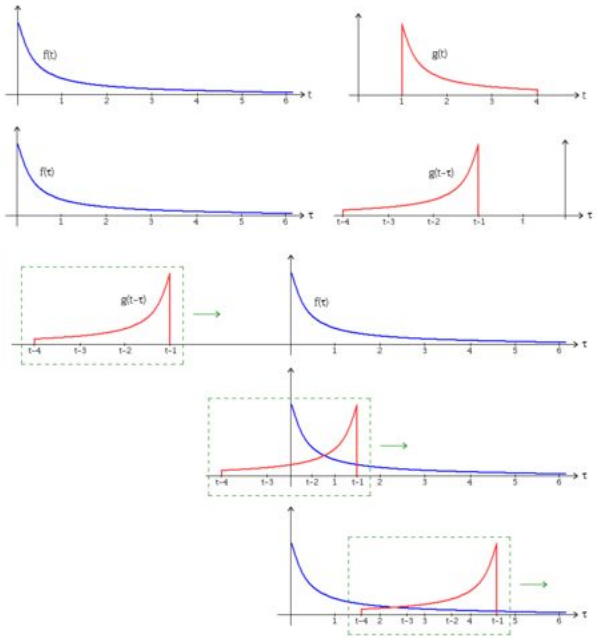
Funcionamiento de las capas de una red neuronal convolucional

La corteza visual primaria (V1) es el área visual que está localizada en el polo posterior de la corteza occipital. Es el área especializada en el procesamiento de información acerca de los objetivos estáticos y en movimiento y es excelente en el manejo de reconocimiento de patrones.

Los primeros vistazos a las redes neuronales artificiales para reconocimiento de imágenes se vieron presentes en 1980 cuando Kunihiko Fukushima expuso su Neocognitron, red neuronal jerárquica de varias capas. Su modelo sería mejorado por Yann LeCun en 1998 añadiendo a la red un método de aprendizaje basado en backpropagation para entrenar el sistema. En 2012 estas redes fueron implementadas para GPU (graphics processing unit) consiguiendo resultados que superaron las expectativas.
En general las redes convolucionales constan de 3 capas :

1. Una capa convolucional.
2. Una capa de reducción (pooling) que se encarga de reducir la cantidad de parámetros al quedarse con los más comunes.
3. Una última capa clasificadora que se encarga de dar el resultado final de la red

La capa convolucional como su nombre lo indica se centra en una operación llamada convolución



Proceso de convolución de 2 funciones

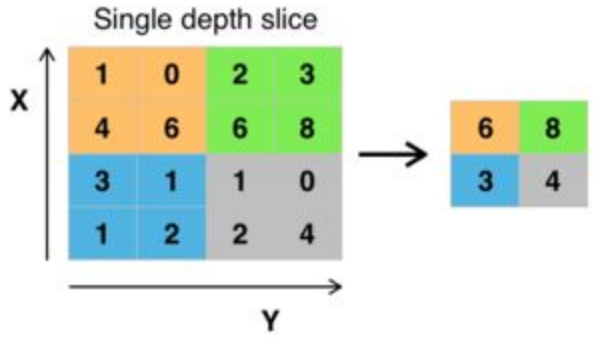
que en matemáticas es conocido como un operador que transforma 2 funciones en una tercera, que en cierto sentido representa la magnitud en la que se superponen la primera función y una versión trasladada e invertida de la segunda función. Para las CNN (convolutional neural networks) la convolución actúa recibiendo una imagen 2D de entrada, para el cálculo de la salida se dispone de la función:

$$Y_j = g\left(b_j + \sum_i K_{ij} \otimes Y_i\right)$$

Donde Donde la salida Yj de una neurona j es una matriz que se calcula por medio de la combinación lineal de las salidas Yi de las neuronas en la capa anterior cada una de ellas operadas con el núcleo de convolucional Kij correspondiente a esa conexión. Esta cantidad es sumada a una influencia bj y luego se pasa por una función de activación no-lineal.

Al pasar por esta capa y ser trabajada por la función de las neuronas de convolución la imagen se filtra ya que ciertas características se vuelven más dominantes en la imagen de salida al tener éstas un valor numérico más alto asignados a los píxeles que las representan.

La capa de pooling se encarga de reducir las dimensiones de la imagen esto en términos del ancho y el alto de la imagen sin que se vea afectada la profundidad de esta. Esta capa también es llamada de reducción de muestreo, ya que la reducción de las medidas de la imagen también conlleva a la pérdida de información, pero para este caso resulta útil, ya que permite centrarse en las principales características permitiendo así una reducción en el cálculo que requieren las otras capas.
Esta capa lleva cabo su trabajo por una operación llamada max-pooling que es un proceso de discretización basado en muestras. Este proceso elige la agrupación máxima de cada subregión, que se realiza aplicando un filtro máximo a estas subregiones no superpuestas de la representación inicial.



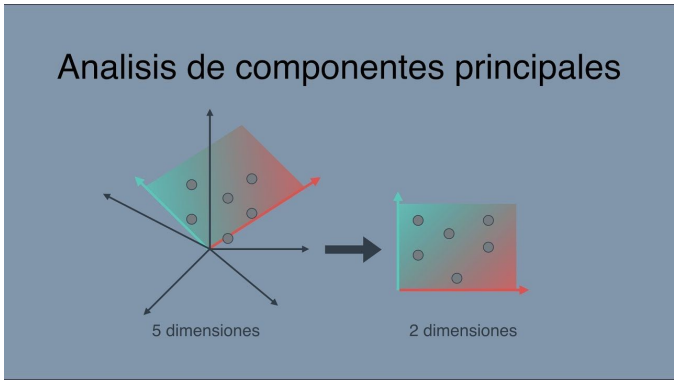
Operación max-pooling

Por último la capa de clasificación se evidencia como varias capas completamente conectados en la que cada píxel se considera como una neurona separada al igual que en una red neuronal regular.

Para el manejo de estas capas se usó la librería Keras que es una librería de redes neuronales de código abierto escrita en python.

B. EXTRACCIÓN DE CARACTERÍSTICAS PRINCIPALES

El análisis de componentes o PCA por sus siglas en inglés es una técnica que se usa para describir un conjunto de datos en términos de nuevas variables no correlacionadas. Esta técnica es útil para reducir la dimensionalidad de un conjunto de datos, por ello es usada en el análisis exploratorio de datos y para la construcción de modelos predictivos.



Uso de APC para reducir dimensionalidad del conjunto de datos

Hay 2 formas básicas de PCA:

1. Método basado en correlaciones.
2. Método basado en covarianzas.

El método basado en correlaciones parte de la matriz de correlaciones, se considera el valor de cada una de las m variables aleatorias Fj. Para cada uno de los n individuos tomemos el valor de estas variables y escribamos el conjunto de datos en forma de matriz:

$$(F_j^\beta)_{j=1,\dots,m}^{\beta=1,\dots,n}$$

donde se observa que cada conjunto

$$\mathcal{M}_j = \{F_j^\beta | \beta = 1, \dots, n\}$$

puede considerarse una muestra aleatoria para la variable Fj. A partir de los m*n datos correspondientes a las m variables aleatorias, puede construirse la matriz de correlación muestral, que viene definida por:

$$\mathbf{R} = [r_{ij}] \in M_{m\times m} \qquad \text{donde} \qquad r_{ij} = \frac{\operatorname{cov}(F_i, F_j)}{\sqrt{\operatorname{var}(F_i)\operatorname{var}(F_j)}}$$

Puesto que la matriz de correlaciones es simétrica entonces resulta diagonalizable y sus valores propios λi verifican:

$$\sum_{i=1}^{m} \lambda_i = m$$

Debido a la propiedad anterior estos m valores propios reciben el nombre de pesos de cada uno de los m componentes principales. Los factores principales identificados matemáticamente se representan por la base de vectores propios de la matriz R. Está claro que cada una de las variables puede ser expresada como combinación lineal de los vectores propios o componentes principales.

Para el método de covarianza el objetivo es transformar un conjunto dado de datos X de dimensión n x*m a otro conjunto de datos Y de menor dimensión n x l con la menor pérdida de información útil posible utilizando para ello la matriz de covarianza.

Se parte de un conjunto n de muestras cada una de las cuales tiene m variables que las describen y el objetivo es que, cada una de esas muestras, se describa con solo I variables, donde I < m. Además, el número de componentes principales I tiene que ser inferior a la menor de las dimensiones de X.

$$l \leq \min\{n, m\}$$

Los datos para el análisis tienen que estar centrados a media 0 y/o auto escalados.

$$\mathbf{X} = \sum_{a=1}^{l} \mathbf{t}_a \mathbf{p}_a^T + \mathbf{E}$$

Los vectores **t**a son conocidos como scores y contienen la información de cómo las muestras están relacionadas unas con otras además, tienen la propiedad de ser ortogonales. Los vectores pa se llaman loadings e informan de la relación existente entre las variables y tienen la cualidad de ser ortonormales. Al coger menos componentes principales que variables y debido al error de ajuste del modelo con los datos, se produce un error que se acumula en la matriz E.

El PCA se basa en la descomposición en vectores propios de la matriz de covarianza. La cual se calcula con la siguiente ecuación:

$$\operatorname{cov}(X) = \frac{X^T X}{n-1}$$

$$\operatorname{cov}(X)\ \mathbf{p}_a = \lambda_a\ \mathbf{p}_a$$

$$\sum_{a=1}^{m} \lambda_a = 1$$

Donde λa es el valor propio asociado al vector propio **p**a. Por último,

$$\mathbf{t}_a = X\,\mathbf{p}_a$$

Esta ecuación se puede entender como que **t**a son las proyecciones de X en **p**a, donde los valores propios λa miden la cantidad de varianza capturada, es decir, la información que representan cada uno de los componentes principales. La cantidad de información que captura cada componente principal va disminuyendo según su número es decir, el componente principal número uno representa más información que el dos y así sucesivamente.

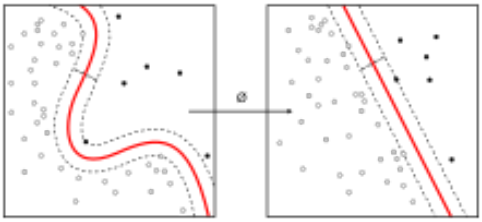
este modelo está limitado ya que presupone que los datos están presentados como una combinación lineal de una base específica.

C. MÁQUINAS DE VECTORES DE SOPORTE

Las máquinas de vectores de soporte son un modelo donde la función objetivo a minimizar es convexa, haciéndolas mucho más fácil de entrenar que las redes neuronales, estas son usadas para la clasificación y regresión en aprendizaje supervisado, aunque también puede ser usado para clustering en aprendizaje no supervisado.

Como muchos otros modelos, las máquinas de vectores de soporte construyen un conjunto de hiperplanos con el fin de separar las clases que se entregaron.

Comúnmente, las clases no son linealmente separables en el espacio de entrada, para ello, las SVM hacen uso de funciones kernel que logran separar los datos en otro espacio.



Transformación de los datos del espacio de entrada al espacio característico a través de la función kernel.

Antes de entrar en detalle se debe hablar sobre las máquinas basadas en kernel.

Máquinas Basadas en Kernel

Para hablar de las máquinas basadas en kernel, debemos recordar que la salida de una red neuronal está dada por

$$z_j = h(a_j).$$

Donde a_j pasa por una función de activación no lineal, esta función normalmente está entre las funciones: tangente hiperbólica, sigmoide, exponencial, relu, etc.

Si se reemplaza la función h por una **función kernel k**, el modelo se conocerá como una máquina basada en kernel y quedará de la forma

$$y(\mathbf{x}) = \sum_{n=1}^{N} w_n k(\mathbf{x}, \mathbf{x}_n) + w_0$$

Las funciones kernel son parecidas a las funciones de activación pero con ciertas diferencias (su función de separar los datos en un espacio característico es la misma, pero con diferente forma de realizarlo), algunas características de las funciones kernel son:

- Es una función real de dos argumentos
- Son funciones simétricas, donde k(x, x') = k(x', x)
- Son funciones no negativas, k(x, x') siempre será mayor a cero

Por lo tanto, funciones como tangente hiperbólica quedan fuera de las funciones kernel.

Las funciones kernel también son concebidas como funciones de similaridad, las cuales retornan qué tan similares son sus entradas X y X'

Sparse Kernel Machines

Uno de los objetivos de las máquinas basadas en kernel, es introducir sparcity, esto significa que un gran número de pesos sean convertidos en ceros, para así lograr que el modelo dependa de muy pocos parámetros, de este tema se extrae otra ventaja de las SVM en relación a las redes neuronales, ya que muchas veces se requiere que M (número de muestras) sea mucho mayor que D (número de parámetros del modelo), esto requiere que el conjunto de datos de la red neuronal sea muy grande, ya que el número de parámetros de esta crece exponencialmente a través de las capas.

Entonces, al introducir sparcity, el modelo depende de muy pocos datos, estos datos son conocidos como **vectores de soporte**, de ahí viene el nombre del modelo.

Una forma de introducir sparcity es usando la siguiente regularización

$$J_2(\mathbf{w}) = L(\mathbf{t}, \mathbf{y}) + \frac{\lambda}{2}\sum_{j=0}^{N} w_j^2$$

Donde L(t, y) es una función de pérdida, la cual también se debe modificar.

Funciones Kernel

Las funciones kernel son el corazón de las SVM, en este modelo, se reemplaza la función de error por un kernel. Algunas funciones kernel son:

Kernel exponencial cuadrático:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{x}')^{\top}\Sigma^{-1}(\mathbf{x}-\mathbf{x}')\right\}$$

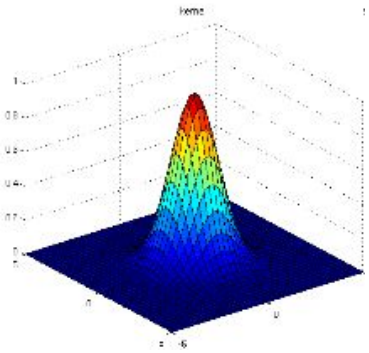


Imagen n. Kernel exponencial cuadrático en 2D

Donde ∑ es la matriz de covarianza, la cual puede ser diagonal o esférica.

Para casa tipo de problema se tiene un tipo de kernel, los cuales se mencionan a continuación

- Para texto se usan los *string kernels*
- Para imágenes se usan los *Pyramid match kernels*
- Para modelos probabilísticos se usan los *Fisher Kernels*

Función objetivo de las SVM

Ya habiendo visto la teoría de la SVM y las funciones ketrnel, podemos hablar sobre la **función objetivo a minimizar**, la cual es convexa, lo que significa que solo tiene un mínimo, que será el mínimo global, por lo tanto, a las SVM se le puede aplicar cualquier tipo de optimización convexa con satisfacción de restricciones, como simplex o la gran M.

La función objetivo de la SVM está dada por la siguiente expresión

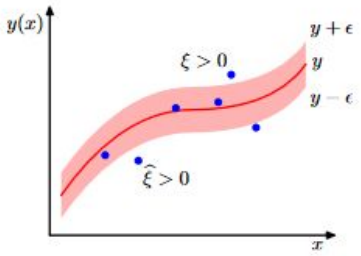
$$J(\mathbf{w}) = C\sum_{n=1}^{N} L_{\epsilon}(t_n, y_n) + \frac{1}{2}\|\mathbf{w}\|_2^2,$$

Donde W son los pesos del modelo, 't' son las etiquetas a predecir, 'y' es la salida de la SVM y C es una constante de regularización, que también se conoce como la penalización del error.

L es la función de pérdida, que está dada por la expresión

$$L_{\epsilon}(t, y) = \begin{cases} 0, & \text{si } |t-y| < \epsilon \\ |t-y| - \epsilon, & \text{de otra forma.} \end{cases}$$

ϵ Es el umbral de error, gráficamente pude verse de la siguient manera



La línea roja es t, y la parte sombreada es el umbral permitido.

Por lo tanto, a partir de este umbral se introducen las variables ξ y ξ', que serán variables de restricciones, estas variables están asociadas a cada dato

- $\xi > 0$ será cuando $t_n > y(x_n) + \epsilon$
- $\xi' > 0$ será cuando $t_n < y(x_n) - \epsilon$

Todos los puntos bien clasificados deben estar dentro del tubo, estas restricciones son de la forma

$$t_n \leq y(\mathbf{x}_n) + \epsilon + \xi_n$$
$$t_n \geq y(\mathbf{x}_n) - \epsilon - \hat{\xi}_n.$$

Por lo que, solo se tendrán en cuenta para el error los puntos fuera del tubo, estos son los que se deben arreglar, entonces, es posible escribir una función la cual deba ser minimizada solo con estos puntos.

$$C\sum_{n=1}^{N}(\xi_n+\hat{\xi}_n)+\frac{1}{2}\|\mathbf{w}\|_2^2,$$

Esta función está sujeta a las siguientes restricciones

$$\xi_n \geq 0$$
$$\hat{\xi}_n \geq 0$$
$$t_n \leq y(\mathbf{x}_n)+\epsilon+\xi_n$$
$$t_n \geq y(\mathbf{x}_n)-\epsilon-\hat{\xi}_n.$$

Para minimizar esta ecuación se puede usar cualquier método de optimización convexa, como multiplicadores de Lagrange.

Podemos observar que, a diferencia de otros modelos, la minimización del error en las SVM tiene una complejidad extremadamente baja, por lo que es muy recomendable su uso.

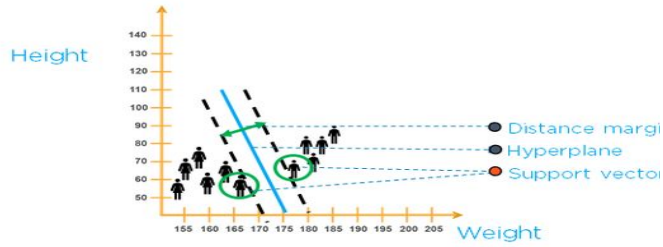
Predicción con las SVM

La función de predicción es de la forma

$$y(\mathbf{x})=\sum_{n=1}^{N}(a_n-\hat{a}_n)k(\mathbf{x},\mathbf{x}_n)+w_0.$$

Donde los vectores de soporte son los puntos que están en los límites del tubo, gráficamente, serán los puntos que definirán el hiperplano entre las clases.

En clasificación, un concepto muy importante es el concepto de **margen**, este es la distancia más pequeña entre la frontera de decisión y las muestras



D. EXPLICACIÓN DEL CÓDIGO Y RESULTADOS

Antes de empezar con la explicación del código, cabe destacar que el desempeño varía cada vez que se ejecuta el programa a pesar de tener semillas definidas y random_state, pero los resultados siguen siendo muy similares y por lo tanto, pueden ser usados para el análisis.

Carga de dataset y liberías

Se usaron elementos de 3 librerías diferentes, de las cuales se sacaron varios elementos de sklearn: el dataset fetch_lfw_people del elemento de la libería sklearn.datasets, la libería NumPy para el manejo de arreglos más sofisticados en el programa, de la librería matplotlib, el elemento pyplot para el manejo de figuras simulando el funcionamiento de MATLAB, de la librería sklearn.model_selection se importaron GridSearchCV y train_test_split los cuales sirven para implementar un método de ajuste y puntuación y para dividir matrices en subsets aleatorios de prueba y entrenamiento respectivamente, los elementos accuracy_score y confusion_matrix de la librería sklearn.metrics que son de utilidad para la exactitud de la clasificación de los subsets y por último, la librería sklearn.decomposition para la reducción de dimensiones lineal.

Se empieza asignando elementos del datset a variables independientes en el programa y después creando una función para el almacenamiento de las imágenes de las caras que se van a usar, el resultado de este proceso es usado por todas las formas de clasificar rostros en este programa (Vectores de soporte y redes neuronales).

1. Redes Neuronales.

Se hizo una prueba de cinco distintas configuraciones de hiperparámetros y se importaron 3 elementos de 2 librerías:

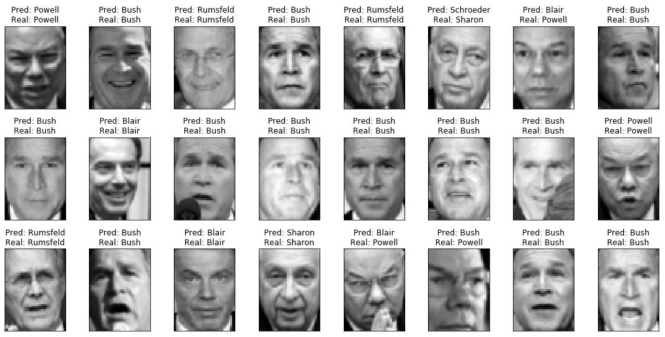
De sklearn.neural_network se importó MLPClassifier, el cual es un clasificador de perceptrón multicapa; se usó la librería tensorflow.keras, de la que se importaron dos elementos: layers y Model, que ayudan a construir y entrenar modelos.

Primera aproximación:

La primera aproximación se hace sin extraer características a través de capas de convolución y pooling. Aunque no se extraigan a través de convolución, son extraídas a través de decomposition (a lo que se llamará desde ahora PCA).

En esta aproximación es posible variar los hiperparámetros de una manera sencilla a partir de la utilidad 'GridSearchCV', la cual realiza todas las combinaciones de hiperparámetros dados en el diccionario param_grid.

Esto ocasiona que el tiempo de ejecución sea muy alto, pero también podrá entregar así la mejor combinación, ya que se prueban muchas arquitecturas para la red. Anteriormente se tenía un mayor número de valores para los hiperparámetros, pero el tiempo de ejecución se volvía inaceptable para realizar las pruebas necesarias.



Antes de realizar PCA, se realizó la estimación la cual dio un accuracy de 82%, sin embargo, luego de agregar la extracción de características bajó de manera sustancial, esto era de esperarse pues se pasó de 1850 características a menos de 300. Cuando se incrementa el número de componentes, se puede ver un incremento en el accuracy, siendo así directamente proporcional hasta 220 características, luego de esto, el accuracy comienza a decrementar, es de importancia recalcar que, aunque se haya fijado un random_state, los resultados varían, por lo que la disminución luego de 220 características puede no darse en el momento de ejecutar. La disminución del accuracy puede deberse a que se están tomando características que no son relevantes en el reconocimiento del rostro.

A continuación se muestra una tabla en la cual se registra los efectos de variar el número de componentes principales a seleccionar.

Número de componentes	Accuracy con el mejor estimador
150	0.70155
170	0.71705
190	0.736434
220	0.763565
250	0.751937
280	0.705426

Segunda aproximación:

Se realiza una segunda aproximación con 3 capas de convolución y pooling y una capa densa de 512 neuronas.

Primero se hace la arquitectura del modelo definiendo el tamaño de la imagen, haciendo 3 veces 16 filtros de 3x3 con un pooling de 2x2 y al final se compila el modelo.

Después se hace un entrenamiento del modelo usando de la librería keras.utils el elemento to_categorical. Finalizado el entrenamiento, se hace la predicción y después el testing.

Para esta arquitectura, en 30 épocas se obtuvo un Accuracy en el entrenamiento de 92.02% y en el testing de 85.6589%, los cuales pueden ser considerados unos buenos resultados.

No es comparable este modelo (y los siguientes con capas de convolución) con el modelo inicial ya que en este se cuenta con muchos más parámetros que el anterior, donde, a la capa de la red neuronal normal luego de las capas de convolución y

pooling le ingresan más de 18 mil parámetros, a diferencia del otro donde inicia con alrededor de 200 parámetros.

Tercera aproximación:

Se realiza una tercera aproximación con 2 capas de convolución y una capa densa de 1024 neuronas.

Se realiza la arquitectura y el entrenamiento del modelo teniendo en cuenta de que se modifica el valor de Dropout, lo que afecta el overfitting.

Después del entrenamiento viene la predicción y el testing y al final se obtiene que, en 27 épocas, se obtuvo un Accuracy en el entrenamiento de 99.71% y en el testing, de 83.7209%, los cuales se pueden considerar buenos resultados.

Cuarta aproximación:

Por último, se realiza una cuarta aproximación modificando el número de capas densas y el número de neuronas en cada capa.

Se usan la librería tensorflow.keras.optimizers para importar el elemento RMSprop y se definen las semillas de la librería numpy para la aleatoriedad. Se hace la arquitectura y el entrenamiento correspondiente.

Al final se hace la predicción y el testing, dando como resultado, en 30 épocas, un Accuracy en el entrenamiento de 90.58% y en el testing de 81.3953%. A pesar de que son muy buenos resultados, se ve un decremento del desempeño en el accuracy de ambos casos.

Análisis global del uso de Redes Neuronales.

Se puede evidenciar que los modelos con capas de convolución y pooling obtuvieron un desempeño mejor que el modelo inicial de extracción de características, como se mencionó anteriormente, esto se debe a que cuentan con una mayor cantidad de parámetros los cuales facilitan la predicción de los rostros.

A continuación se muestra un reporte de los accuracy obtenidos en los tres acercamientos.

Modelo	Características	Accuracy

1	Extracción de características con PCA
	# Características = 220
	Alpha = 0.001
	Hidden Layer Sizes = (512)
	Learning rate Init = 0.001
		0.763565

2	Extracción de características con capas de convolución 3 Capas de convolución y pooling Una capa densa de 512 neuronas	0.856589
3	Extracción de caracaterísticas con capas de convolución 2 Capas de convolución y pooling Una capa densa de 1024 neuronas	0.837209
4	Extracción de características con capas de convolución 3 Capas de convolución y pooling 4 capas densas de 64, 64, 128 y 128 neuronas respectivamente	0.813953

Se puede inferir de los resultados que, para este problema, el mejor modelo fue el segundo, con solo una capa densa de 512 neuronas y 3 capas de convolución y pooling, se tenía una hipótesis la cual era que entre más capas mejor sería el desempeño, a partir del resultado, la hipótesis fue descartada.

2. Máquinas de vectores de soporte.

Gracias a GrideSearchCV, se puede realizar la variación de parámetros sin la necesidad de escribir mucho código. Partiendo del siguiente resultado del PCA:

El programa encuentra el mejor estimador y realiza la predicción.

A continuación se muestra una tabla que compila los resultados de los hiperparámetros y el accuracy al variar el número de componentes a extraer a partir de PCA.

Modelo	N_components	Características	Accuracy
1	20	Kernel = rbf C = 1 $\gamma = 0.1$	0.697674
2	50	Kernel = rbf C = 1000 $\gamma = 0.01$	0.837209
3	100	Kernel = rbf C = 1000 $\gamma = 0.005$	0.852713
4	150	Kernel = rbf C = 1000 $\gamma = 0.005$	0.82170
5	280	Kernel = sigmoide C = 1 $\gamma = 0.005$	0.70155
6	300	Kernel = sigmoide C = 1 $\gamma = 0.005$	0.68217

En estos resultados se pueden evidenciar varios elementos muy interesantes:

- Podría pensarse que los hiperparámetros del mejor }estimador siempre son los mismos sin importar el número de características que se extraigan a través de PCA, sin embargo, con los resultados obtenidos se puede evidenciar que tanto la función kernel como C y gamma cambian dependiendo de los componentes.
- Podría pensarse que entre más componentes se extraigan, mejor será el desempeño, no obstante, con los resultados se puede observar que luego de 100 parámetros, el desempeño comienza a disminuir. Esto puede deberse a que,para el caso de más de 150 componentes, se están tomando en cuenta componentes que no son relevantes para definir de quién es el rostro, sino que estos solo están aportando ruido. Y que para el caso de menos de 50 componentes, no se tiene el número de características suficientes para poder diferenciar algunas caras.

A partir de los resultados, puede concluirse que para este caso, el mejor modelo para clasificar los rostros es el modelo 3, con 100 características.

Comparación de Redes Neuronales con Máquinas de vectores de soporte.

A continuación se muestra una tabla con el mejor resultado de accuracy para la red neuronal y para la SVM:

Modelo	Accuracy
Red Neuronal	0.856589
SVM	0.852713

Se puede evidenciar que ambos desempeños son muy parecidos, pero en su arquitectura son muy diferentes.

El modelo de la red neuronal con el que se obtuvo este desempeño fue una red con varias capas de convolución y pooling, y luego una capa densa de 512 neuronas, esto da como resultado 289,543 parámetros, lo que, en tiempo de cómputo se ve reflejado, en cambio, para la SVM, se tienen solo 100 parámetros, esto es una diferencia significante.

Si se desea realizar pruebas a partir de una base de datos, y maximizar el accuracy lo más posible no es una prioridad, será mejor usar una SVM, ya que tiene muchos menos parámetros, lo cual también repercute en que no se necesita un dataset tan grande. Pero si se desea tener un accuracy muy elevado, se pueden probar diversas arquitecturas de una red neuronal, la cual tendrá muchos más parámetros pero comúnmente se tendrá un mejor desempeño.

III. CONCLUSIONES

El modelo de Red Neuronal es muy útil cuando la prioridad es tener el mejor accuracy posible, pero al hacer esto también se necesita un dataset mucho más grande. Si por el contrario, no es muy prioritario tener un muy alto accuracy, es mejor usar las Máquinas de vectores de soporte, debido a que utilizan muchos menos parámetros y no requieren un dataset tan amplio para dar una predicción adecuada.

El desarrollo del proyecto demostró que la hipótesis de que, mientras más capas se tuvieran en una red neuronal mejor sería su desempeño, es erróneo.

REFERENCIAS

[1] https://en.wikipedia.org/wiki/Support-vector_machine#Definition
[2] https://www.cs.toronto.edu/~duvenaud/cookbook/
[3] https://es.wikipedia.org/wiki/Corteza_visual#Corteza_visual_primaria_(V1)
[4] https://en.wikipedia.org/wiki/Neocognitron
[5] https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales
[6] https://relopezbriega.github.io/blog/2016/08/02/redes-neuronales-convolucionales-con-tensorflow/

[7] https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks
[8] https://es.wikipedia.org/wiki/An%C3%A1lisis_de_componentes_principales
[9] https://www.tensorflow.org/guide/keras/overview

[10] https://scikit-learn.org/stable/