PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Escuela de Posgrado

Redes neuronales convolucionales para datos composicionales: Una aplicación a la industria textil de la moda

Tesis para obtener el grado académico de Magíster en Estadística que presenta:

Pavel Arturo Cotacallapa Amanqui

Asesor:

Dr. Luis Enrique Benites Sanchez

Lima, 2022

Dedicatoria

Este trabajo se lo dedico a mi papá Arturo quien me guía desde el cielo. Cuando yo nací, él había ingresado a la universidad para estudiar estadística pero nunca pudo terminar esta carrera. Este esfuerzo se lo dedico a él. A mi mamá Mercedes que siempre me cuida y me proteje con su cariño y su bondad, si en este mundo habrían más personas como ella, viviríamos en un mundo muchísimo mejor. A mis hermanos Boris, Ebner y Franco y a mi sobrina Leyla, porque siempre aprendo de ellos más de lo que ellos aprenden de mí.

A mis abuelitos y tíos en el cielo, que fueron el comienzo de todo. A doña Machi, porque hace muchos años le hice una promesa, que si un día publicaba un libro, se lo dedicaría, este trabajo es, de alguna manera, un libro, a ella por ser un ángel en la tierra. Finalmente, y de manera muy utópica, a las diferentes versiones y etapas que de mí mismo existieron en el pasado, porque si mis yos del pasado se hubieran rendido ante cada dificultad simplemente no podría estar dando este paso tan trascendental en mi vida.

Agradecimentos

Tengo mucho que agrader a los profesores de la Pontificia Universidad Católica del Perú, a mi asesor Luis y a todos los compañeros con los que alguna vez compartí clases. Asimismo, a todos los colegas, excolegas y jefes con los que compartí proyectos en los distintos lugares donde trabajé. Aprender de ellos e inspirarme de sus conocimientos y de sus valores positivos, ha marcado en mí un antes y un después como profesional y como persona.

También, y muy especialmente, le agradezco a Dios, a Jesús, a la Virgen María y a los apóstoles San Judas Tadeo y San Mateo, quienes siempre me bendicen y cuidan de mi y de mi familia desde su santidad y omnipotencia, porque cada día que pasa me ayudan a transformar mi corazón y hacen que, por medio de mis reflexiones y de mis actos, sea una mejor persona, y porque gracias a su infita grandeza han permitido que la humanidad vaya descubriendo, por medio de la ciencia, todas sus bellas creaciones.

Resumen

En muchas situaciones prácticas es necesario el uso de modelos que puedan predecir una colección de datos limitados por un intervalo cuya suma sea una constante por cada unidad estadística. Este tipo de variable respuesta se conoce como datos composicionales. Por otro lado, el número de covariables que se usan para el entrenamiento de este tipo de modelos pueden provenir de datos asociados a imágenes como la intensidad de los pixeles. En ese contexto, se propone el uso de las redes neuronales convolucionales como una primera alternativa para intentar estimar este tipo de variable respuesta. Se utiliza la distribución de Dirichlet como distribución condicional de los datos y finalmente se propone una aplicación del modelo utilizando imágenes de prendas de vestir que se venden por catálogo en donde el objetivo es predecir las participaciones de las tallas que se venden por cada unidad estadística.

Palabras-clave: Redes Neuronales Convolucionales, Distribución de Dirichlet, Datos Composicionales.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Consideraciones Preliminares

La compra satisfactoria de una prenda de vestir en el mercado de comercio en línea está fuertemente ligada con la elección correcta de la talla. Luce (2019) muestra que en el año 2016 se devolvieron alrededor de sesenta y dos mil millones de dólares en prendas debido a la incorrecta elección de la talla por parte de los consumidores.

Para abordar este problema Song et al. (2017) desarrollan un modelo en el que proponen identificar la talla de determinada persona utilizando como inputs una foto del consumidor que muestre la silueta del individuo. Se identifican visualmente un conjunto de veinte contornos críticos como la cintura, las mangas, el tórax, etc. y se calculan las dimensiones de cada contorno utilizando la imagen que se le ha tomado para posteriormente emplear los pixeles de la imagen como covariables aplicando los modelos de Ridge, SVR, Random Forest y Gradient Boosting para predecir la talla. Concluyen que la regresión Ridge es la más apropiada para estimar este tipo de variable respuesta.

Otro trabajo relevante es el desarrollado por Karessli et al. (2019). En el que se plantea un modelo para la determinación de las tallas de acuerdo a una clasificación binaria utilizando información histórica. Se definen dos categorías, una en la que hubo devoluciones por alguna inconformidad y otra categoría en la que los artículos no se devuelven debido a que no tienen ningún problema en su tamaño. El modelo binomial que se utiliza estima puntuaciones o *scores* por cada tipo de producto para determinar su índice de devolución. Posteriormente, por medio de las imágenes de como se comercializan los productos se utilizan modelos de *CNN (Convolutional Neural Networks)* para extraer la estructura troncal de la prenda y determinar el score por medio de las imágenes. A este tipo de metodología se le denomina *teacher-student approach* en el que una arquitectura basada en *CNN* llamado SizeNet, actúa como estudiante, aprende de las imágenes de las prendas gracias a un modelo estadístico que actua como maestro (porque proporciona el *score*). Sus resultados concluyen que las imágenes de los productos llevan información relevante para determinar el score y por lo tanto la devolución y la mejora en la determinación de la talla.

En el trabajo desarrollado por Guigourès et al. (2018) introducen un enfoque bayesiano para la construcción de un modelo que pueda recomendar la talla de determinada prenda en el comercio electrónico. Se utilizan datos de devoluciones en donde se establecen tres categorías: sin retorno, retorno por ser muy pequeño y retorno por ser muy grande. Se plantea el uso de una distribución

multinomial. Se establecen prioris jerárquicas gracias a la cual se introducen conocimiento y experiencia de las características del artículo. Se muestra que los resultados bayesianos son mejores a los de modelos tradicionales.

Es importante recalcar que los trabajos que se han venido desarrollando enfocan la predicción para la toma de decisión del consumidor, es decir, en predecir la talla que al consumidor le ajustará de manera adecuada. Sin embargo, el enfoque de este trabajo se direcciona en predecir los porcentajes que se demandan por cada talla de determinada prenda de vestir. Para lograr este propósito se utiliza la imagen de cómo se comercializa el producto. Se utilizan las matrices RGB *Red Green Blue* para construir las covariables y entrenar el modelo propuesto con una distribución condicional Dirichlet. Es decir, tiene un enfoque global de las tallas y no de recomendación individual para cada tipo de consumidor.

Por otro lado, la metodología que se propone ha sido estudiada por Bishop (1994) en la que a fin de obtener una descripción completa de los datos predichos se modela la distribución de probabilidad condicional de los datos objetivos condicionados con el vector de entrada. En esta medida se introduce una nueva clase de modelos de redes neuronales obtenidos mediante la combinación de una red neuronal convencional junto con un modelo de distribuciones mixturas. Este tipo de metodología permite, por ejemplo, establecer intervalos de confianza para los datos predichos, poder identificar los momentos y también permite usar otras estadísticas distintas a la esperanza condicional como valores predictivos.

También es importante considerar el trabajo de Wu et al. (2019) que utiliza mixtura de distribuciones Dirichlet acoplandola a una red neuronal convolucional para poder evaluar el riesgo en un problema de clasificación. Este método de ensamblaje logra una estructura muy útil en el proceso de inferencia para datos que puedan soportarse en un simplejo.

1.2. Objetivos

El objetivo general de este trabajo es desarrollar un modelo de redes neuronales convolucionales desde el punto de vista teórico para su posterior aplicación en la estimación de las participaciones de las tallas de prendas de vestir utilizando los pixeles de la imagen del producto como covariables.

Los objetivos específicos alineados a este trabajo son los siguientes:

- Revisar conceptos preliminares como el modelo de redes neuronales, el operador convolucional, *pooling* y el modelo de redes neuronales convolucionales en general.
- Desarrollar un modelo teórico para datos composicionales con redes neuronales convolucionales utilizando la distribución Dirichlet como la distribución condicional de la variable respuesta.
- Evaluar los criterios de selección de un modelo de esta naturaleza.
- Desarrollar intervalos de confianza para cada elemento de la variable respuesta, es decir, de los datos composicionales.

- Aplicar el modelo para un conjunto de imágenes de prendas para la determinación de la participación de las tallas por cada tipo de producto.

1.3. Organización del Trabajo

En el Capítulo 2 se desarrollan conceptos previos y básicos que serán utilizados en la construcción del modelo final. En el Capítulo 3 se construye el modelo de redes neuronales convolucionales utilizando la distribución Dirichlet como distribución condicional de la variable respuesta. Se introduce la función negativa de la log-verosimulitud de la variable respuesta como función de costo para poder estimar los parámetros de la distribución por cada unidad estadística. En el Capítulo 4 se realiza un estudio de simulación que tiene por objetivo de mostrar el desempeño del modelo que se propone considerando escenarios diferenciados. En el Capítulo 5 se aplica el modelo a un conjunto de datos reales. Se busca estimar las participaciones o porcentajes en las tallas de prendas de vestir utilizando los datos de los pixeles de la imagen del producto como datos de entrada. En el Capítulo 6 se discuten las conclusiones que se obtienen en el desarrollo de este trabajo.

En los anexos se presentan los códigos de programación que se usan para estimar el modelo en python.

Capítulo 2

Conceptos Básicos

En este capítulo se presentan los conceptos más importantes relacionados con este trabajo así como la distribución Dirichlet relacionada a la construcción del modelo que se desarrolla en el siguiente capítulo.

2.1. Datos Composicionales

Un conjunto de datos es denominado composicional cuando los elementos de los que está compuesto son no negativos y su suma es igual a uno como lo define Pawlowsky-Glahn y Buccianti (2011). Matemáticamente, los datos composicionales pueden ser representados por puntos en un simplejo. Un ejemplo de dichos datos son las probabilidades, proporciones y porcentajes. Asimismo, Aitchison (1986) define un conjunto de datos composicionales delimitado en el espacio real $D-$dimensional de la siguiente manera:

$$S^D = \{\boldsymbol{Y} = [y_1, y_2, ..., y_D] \in \mathbb{R}^D : y_i > 0,\ \ i = 1, 2, ..., D;\ \ \sum_{i=1}^{D} y_i = 1\}.$$

Es importante señalar que en el contexto que enmarca este trabajo, cada vector de datos composicionales esta asociado a una unidad estadística como variable respuesta. Por otro lado, si en la ecuación anterior *n=3* se puede obtener la siguiente ecuación:

$$S^3 = \{\boldsymbol{Y} = [y_1, y_2, y_3] \in \mathbb{R}^3 : y_i > 0,\ \ i = 1, 2, 3;\ \ y_1 + y_2 + y_3 = 1\}.$$

Si se asumen *n* observaciones entonces se tienen *3n* valores que pueden presentarse en una figura denominada gráfico ternario. La Figura 2.1 muestra la manera de representar un conjunto de datos composicionales de tres dimensiones por medio de un plano euclidiano en el lado izquierdo y un gráfico ternario de dos dimensiones en el lado derecho.

Es fácil observar que los puntos en la representación tridimensional están altamente correlacionadas debido a su naturaleza composicional. El gráfico ternario dibuja los puntos en la intersección de la prolongación de cada uno de los valores de los tres ejes. En el caso de la $Variable\ \ Z$ es una prolongación horizontal mientras que en el caso de la $Variable\ \ X$, una prolongación con inclinación de pendiente negativa y en el caso de la $Variable\ \ Y$ una prolongación con inclinación de

pendiente positiva.

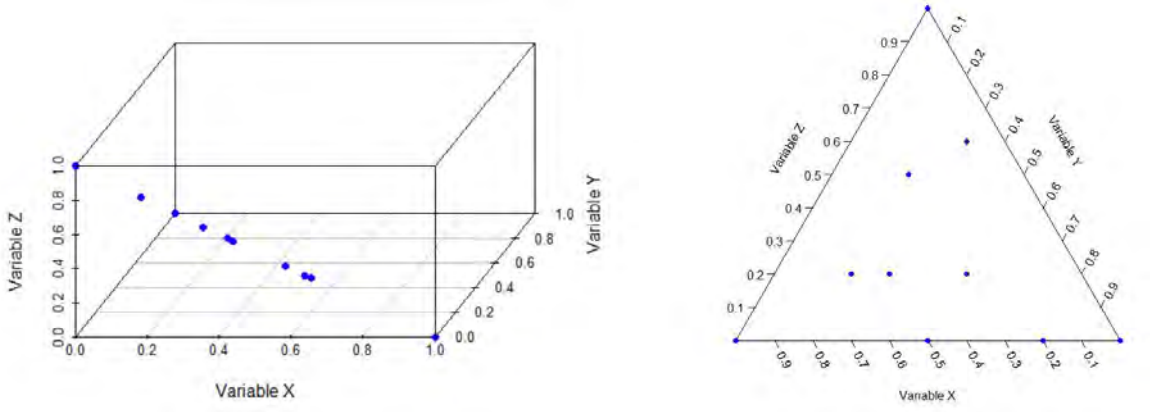

Figura 2.1: Representación tridimensional y ternaria.

2.2. Distribución Dirichlet

La distribución Dirichlet fue desarrollada por el matemático alemán Johann Peter Gustav Lejeune Dirichlet y se denota como $Dir(\boldsymbol{\alpha})$. Pertenece a la familia de distribuciones de probabilidad continuas multivariadas. Los parámetros de esta distribución pertenecen al vector $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, ..., \alpha_K)$ en donde: $\alpha_k > 0$, para cada uno de los elementos del vector $\boldsymbol{\alpha}$. Asimismo, se puede concebir a la distribución Dirichlet como como una generalización multivariada de la distribución beta.

2.2.1. Función de densidad de probabilidad

La función de densidad de probabilidad se expresa de la siguiente manera:

$$f(y_1, y_2, ..., y_K; \boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{k=1}^{K} y_k^{\alpha_k - 1},$$

en donde el vector aleatorio $\boldsymbol{Y} = [y_1, y_2, ..., y_K]$ pertenece al simplejo de orden K y $\alpha_k > 0$, para cada elemento que pertenece al vector $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_K]$.

$$S^{(K)} = \{[y_1, y_2, ..., y_K] \in \mathbb{R}^K : y_k > 0, \;\; k = 1, 2, ..., K; \;\; \sum_{k=1}^{K} y_k = 1\}.$$

Por otro lado, la constante normalizadora es la función multivariada beta que se expresa de la siguiente manera:

$$B(\boldsymbol{\alpha}) = \frac{\prod_{k=1}^{K} \Gamma(\alpha_k)}{\Gamma(\sum_{k=1}^{K} \alpha_k)}.$$

Asimismo, la función gamma se define a continuación:

$$\Gamma(\alpha_k) = \int_0^\infty s^{\alpha_k - 1} e^{-s} ds.$$

Por lo tanto, la función de densidad de probabilidad se puede reescribir:

$$f(y_1, y_2, ..., y_K; \boldsymbol{\alpha}) = \frac{\Gamma(\sum_{k=1}^K \alpha_k)}{\prod_{k=1}^K \Gamma(\alpha_k)} \prod_{k=1}^K y_k^{\alpha_k - 1}.$$

La distribución Dirichlet ha sido ampliamente utilizada para el tratamiento de datos composicionales. Por ejemplo, Maier (2014) utiliza esta distribución como la distribución condicional para abordar un modelo de regresión de datos composicionales. En adelante, se utilizará la notación $\boldsymbol{Y} = [Y_1, Y_2, ..., Y_K] \sim Dir(\boldsymbol{\alpha})$ para referirnos que el vector $\boldsymbol{Y}$ sigue una distribución Dirichlet con un vector de parámetros $\boldsymbol{\alpha}$.

2.2.2. Propiedades

Sea un vector aleatorio $\boldsymbol{Y} = [y_1, y_2, ..., y_K] \sim Dir(\boldsymbol{\alpha})$ entonces la esperanza y varianza de los elementos y la covarianza entre los elementos del vector aleatorio quedan definidos a continuación:

$$E[y_i] = \frac{\alpha_i}{\sum_{k=1}^K \alpha_k},$$

$$Var[y_i] = \frac{\tilde{\alpha_i}(1 - \tilde{\alpha_i})}{\alpha_0 + 1},$$

en donde:

$$\tilde{\alpha_i} = \frac{\alpha_i}{\sum_{k=1}^K \alpha_k},$$

$$\alpha_0 = \sum_{k=1}^K \alpha_k,$$

$$\text{y } Cov[y_i, y_j] = \frac{-\alpha_i \alpha_j}{\alpha_0^2(\alpha_0 + 1)} \ \forall \ i \neq j.$$

2.3. Redes Neuronales

El término redes neuronales cubre un amplio número de modelos diferentes muchos de los cuales han sido objeto de afirmaciones exageradas con respecto a su plausibilidad biológica afirma Bishop (2006). Sin embargo, este trabajo se construye sobre la base de la implicancia de que las redes neuronales sirven para identificar patrones y poder realizar predicciones. Una red neuronal está compuesta por tres elementos globales, datos de entrada, datos de salida y datos o neuronas ocultas que llevan información difícil de interpretar pero que traen consigo múltiple información que será

transferida a la siguiente neurona o finalmente a la variable respuesta. La Figura 2.2 muestra esta estructura. Se puede notar que los datos de entrada componen una capa que se conoce como *input layer*. Las capas intermedias u ocultas se denominan *hidden layers* y finalmente la capa de salida o la variable respuesta se conoce como *output layer*.

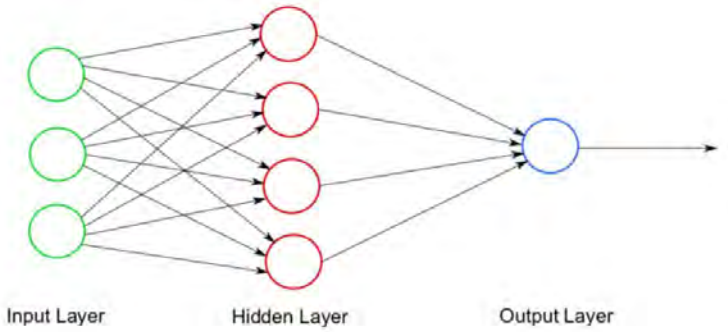

Figura 2.2: Estructura de una red neuronal.

El vector que contiene a los datos de entrada de determinada observación, es decir al *input layer* se puede definir como el vector $\boldsymbol{X}$. En donde $\boldsymbol{X}^T = [x_1, x_2, ..., x_p] \in \mathbb{R}^p$. Es decir, se tienen p covariables con las que se comienza la red. Cada covariable x_i será ajustada multiplicándosele por un peso denotado por w_i para luego ser añadida a alguna de las neuronas de la siguiente capa. Los pesos se pueden representar por la siguiente matriz $\boldsymbol{W}$. Asimismo, $\boldsymbol{W}^T = [w_1, w_2, ..., w_p] \in \mathbb{R}^p$. Por otro lado, a la neurona de la siguiente capa que recepciona los valores de las covariables ajustados por los pesos se le añade un parámetro $b \in \mathbb{R}$. Por lo tanto, determinada neurona de la siguiente capa tendrá la siguiente forma:

$$Z = \boldsymbol{W}^T \boldsymbol{X} + b.$$

Es fácil notar que $Z \in \mathbb{R}$. Del mismo modo, antes de que el valor de la ecuación anterior migre a la siguiente capa se le aplica una función denominada función de activación. La más conocida es la función sigmoidal $\sigma(Z)$:

$$\sigma(Z) = \frac{1}{1 + e^{-Z}} \ , \ \forall \ Z \in \mathbb{R}.$$

Por lo que se puede reescribir la función de la siguiente manera:

$$\sigma(Z) = \sigma(\boldsymbol{W}^T \boldsymbol{X} + b).$$

La ecuación anterior busca predecir la variable respuesta de una observación cuando se llegue a la capa final:

$$\hat{y} = \sigma(Z) = \sigma(\boldsymbol{W}^T \boldsymbol{X} + b),$$

siendo $\hat{y}$ el valor predicho e y el valor real. Con estas consideraciones se construye una función de costos que relacione los valores reales con los predichos:

$$\mathcal{J}(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}_i, y_i).$$

La función de costos más utilizada es la del error cuadrático medio:

$$\mathcal{J}(w,b) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2. \tag{2.1}$$

Reescribiendo la ecuación 2.1 se tiene que:

$$\mathcal{J}(w,b) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \sigma(\boldsymbol{W}^T\boldsymbol{X}_i + b))^2. \tag{2.2}$$

Se busca minimizar la ecuación 2.2 buscando un mínimo global identificando los valores más convenientes del vector de $\boldsymbol{W}$ y b.

2.4. Convolución

La razón por la cual se utiliza la convolución es porque permite reducir el número de parámetros para el entrenamiento de una red neuronal Aghdam y Heravi (2017). El uso de la convolución no es única o nueva para las redes neuronales, por el contrario, ha sido ampliamente utilizado en el campo de *computer vision*. En el marco de este trabajo, se define como operador convolucional a la matriz que actúa como filtro sobre la matriz numérica que representa la imagen de determinado objeto.

Toda imagen está compuesta de un conjunto de pixeles que componen un bloque con un ancho, una altura y una profundidad. Cada elemento, es decir, cada pixel, es un número acotado entre 0 y 1 (o entre los números enteros 0 y 255, dependiendo de la convención que se utilice) lo que significa que se puede representar una imagen con un conjunto de matrices. Asimismo, lo convencional es que cada imagen esté compuesta por tres matrices conocidas en la literatura como RGB (*Red, Greeen and Blue*) como afirma Aghdam y Heravi (2017). En este caso, las tres matrices representan a los colores Red, $Green$ y $Blue$. Por ejemplo, si se tiene una imagen con una dimensión de pixeles de 270 por 180, se puede decir que se está haciendo referencia a tres matrices cuyas dimensiones son de 180 filas por 270 columnas y la profundidad a la que se hacía referencia al inicio del párrafo es de 3 ya que la imagen está compuesta por 3 matrices.

En ese sentido, si se tiene determinada imagen, entonces los elementos que componen la imagen se pueden denotar por la matriz $\boldsymbol{Q} = [\boldsymbol{Q_r}, \boldsymbol{Q_g}, \boldsymbol{Q_b}]$, en donde $\boldsymbol{Q_r}$, $\boldsymbol{Q_g}$ y $\boldsymbol{Q_b}$ son matrices que tienen las mismas dimensiones y cada elemento $q_r, q_g, q_b \in \mathbb{R}$ están acotados entre 0 y 1 y representan a los pixeles de la imagen.

Asumiendo que dicha imagen $\boldsymbol{Q} \in \mathbb{R}^{H\times W\times C}$, el operador convolucional es una función f que actúa como filtro de tal manera que $\boldsymbol{Q} \times f \in \mathbb{R}^{\tilde{H}\times\tilde{W}\times\tilde{C}}$, en donde $\tilde{H}$ y $\tilde{W}$ dependen de la altura y

ancho del filtro que se aplique, mientras que $\tilde{C}$, de la profundidad.

Formalmente, si se tiene una función $f(x, y)$ su función convolucionada es $g(x, y)$ siempre que:

$$g(x, y) = \omega \times f(x, y),$$

en donde:

$$\omega \times f(x, y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx, dy) f(x + dx, y + dy),$$

$$-a \leq dx \leq a \ \ , -b \leq dy \leq b,$$

siendo $f(x, y)$ la imagen original y $g(x, y)$ la imagen filtrada. Además, ω corresponde al operador convolucional o tambien llamado filtro de Kernel.

En un escenario en el que se tiene una función $f \in \mathbb{R}^{h \times w \times c}$, entonces $\boldsymbol{Q} \times f \in \mathbb{R}^{(H-h+1) \times (W-w+1) \times 1}$. Por ejemplo, si el tamaño de una imagen en una sola escala (sea *Red, Green* o *blue*), es de $W \times H$ y se tengan L operadores convolucionales del tamaño $M \times N$, el resultado de la convolución serán L imágenes cuyos tamaños serán matrices de orden $(W - M + 1) \times (H - N + 1)$.

Por ejemplo, si se tiene la matriz $\boldsymbol{Q}$ con las siguientes características:

$$\boldsymbol{Q} = \begin{bmatrix} 5 & 6 & 1 & 3 \\ 3 & 1 & 4 & 3 \\ 1 & 3 & 2 & 4 \\ 5 & 2 & 4 & 9 \end{bmatrix}.$$

Y se utiliza el operador convolucional definido por $\boldsymbol{\Pi}$:

$$\boldsymbol{\Pi} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

Entonces la operación: $\boldsymbol{Q} \times \boldsymbol{\Pi}$ nos generará como resultado la matriz: $\boldsymbol{Q_{convol}}$.

$$\boldsymbol{Q_{convol}} = \begin{bmatrix} 4 & 2 & -2 \\ 0 & -1 & 0 \\ -1 & -1 & -7 \end{bmatrix}.$$

Si se tiene la estructura de colores RGB, es decir, las 3 matrices, el resultado de la aplicación

del operador convolucional a cada parte de las tres matrices con la misma localidad, se suman. En términos de dimensiones, si la imagen cuenta con 80 por 50 pixeles y con la estructura tradicional RGB, se estaría describiendo una dimensión de $80 \times 50 \times 3$. Y si se aplica el operador convolucional con una dimensión de $10 \times 10 \times 3$, en el que se repite el mismo operador convolucional para cada una de las tres capas (RGB), la dimensión nueva será de $71 \times 41 \times 1$, esto es una matriz de 71×41.

2.5. Pooling

Así como el operador convolucional, el propósito del *pooling layer* es reducir el tamaño espacial de la imagen convolucionada seleccionando algunas características locales esenciales como el máximo, mínimo, promedio, etc. La idea principal de *pooling* es considerar una partición del dominio de una función y reemplazar la función en cada elemento de la partición por el más representativo valor de la función en dicho conjunto. Este procedimiento lleva una simple función. Se utiliza una variante bidimensional de agrupación en la construcción de redes neuronales convolucionales. Calin (2019) define el *pooling* en el marco de una sola dimensión.

Sea $f \in [a, b] \to \mathbb{R}$ una función continua y considérese particiones equidistantes del intervalo $[a, b]$ tal que:

$$a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b,$$

en donde el tamaño de la partición, $\frac{b-a}{n}$, es denominada zanco y se denota por:

$$M_i = max \;\; f(x), \;\; \forall \;\; x \in [x_{i-1}, x_i].$$

Asimismo, considérese la siguiente función:

$$S_n(x) = \sum_{i=1}^{n} M_i 1_{[x_{i-1}, x_i)}(x).$$

El proceso de aproximar la función $f(x)$ por la función $S_n(x)$ se llama *max-pooling*.

Utilizando el ejemplo de la Sección 2.4, el valor de la matriz $\boldsymbol{Q}$ luego de pasar por la operación del $max - pooling$ queda definido por $\boldsymbol{Q_{pool}}$:

$$\boldsymbol{Q_{pool}} = \begin{bmatrix} 6 & 4 \\ 5 & 9 \end{bmatrix}.$$

Finalmente, si bien la función max es la más utilizada, también se puede utilizar el promedio simple.

2.6. Padding

En ciertas ocasiones, cuando se aplica el operador convolucional, se utiliza minoritariamente los bordes de la matriz, en dichos casos y cuando la información de los bordes sea muy relevante, se utiliza el $padding$ que se define como el proceso de incrementar el número de filas y columnas en los bordes superiores, inferiores, derecho e izquierdo. Por ejemplo, si se tiene la matriz $\boldsymbol{Q}$ de la Sección 2.4, y se aplica el padding con un valor de 1, la nueva matriz $\boldsymbol{Q_{pad}}$ queda definida de la siguiente manera

$$\boldsymbol{Q_{pad}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 6 & 1 & 3 & 0 \\ 0 & 3 & 1 & 4 & 3 & 0 \\ 0 & 1 & 3 & 2 & 4 & 0 \\ 0 & 5 & 2 & 4 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Es posible utilizar un nivel de $padding$ igual a 2, en cuyo caso aparecerán dos columnas y filas adicionales con los valores de ceros en los bordes de la matriz original, en general se pueden utilizar un $padding$ con un valor de n.

2.7. Stride

El $stride$ es una convención que indica cada cuántas filas o columnas el operador convolucional se desplazará. Es importante destacar que en la Sección 2.4 el operador convolucional se desplazaba en una fila cuando iba de izquierda a derecha y en una fila cuando iba de arriba hacia abajo mientras que en la Sección 2.5 el desplazamiento fue de dos filas y dos columnas y justamente por esa razón la dimensión de la matriz $\boldsymbol{Q_{pool}}$ en el $max - pooling$ quedó más pequeña que la dimensión de la matriz $\boldsymbol{Q_{convol}}$ cuando se le aplicó el operador convolucional. Es así que el valor que se le asigne o especifique al $stride$ terminará ejerciendo un rol importante en cuanto a la reducción del tamaño de la matriz a la que se le está aplicando la operación. Normalmente, el tamaño del $stride$ es de uno cuando se aplica el operador convolucional mientras que es igual al número de filas cuando se trata del $pooling$.

2.8. Redes Neuronales Convolucionales

Las redes neuronales convolucionales han sido ampliamente utilizadas en el procesamiento y búsqueda de patrones por medio de imágenes. La Figura 2.3 muestra la estructura mediante la cual se procesa la información y la estructura de la red neuronal convolucional. La siguiente imagen muestra dicha estructura para un modelo de reconocimiento de dígitos escritos a mano.

Se puede notar que el proceso comienza con la imagen como principal fuente de información. Dicha imagen es sometida a diferentes operadores convolucionales de tamaño 5×5. Posteriormente, a las matrices convolucionadas se les aplica $max - pooling$. Los resultados de dicha operación

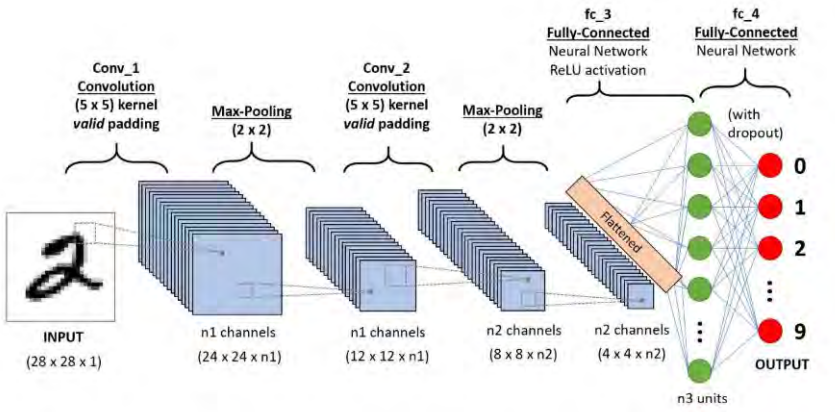

Figura 2.3: Estructura de una red neuronal convolucional. Nota: imagen tomada de towardsdatascience.com.

vuelven a ser sometidos a un proceso convolucional para terminar con otro proceso de $max - pooling$ en donde finalmente se convierten en una vector con una dimensión menor a la imagen original. Dicha información ingresa como datos de entrada en la primera capa y luego comienzan el proceso convencional de las redes neuronales estándar.

Capítulo 3

Modelo de Redes Neuronales Convolucionales para Datos Composicionales

En este capítulo se presenta un modelo de redes neuronales convolucionales utilizando como input inicial las imágenes de determinados productos. La información que lleva consigo las imágenes gracias a los pixeles son procesadas mediante lás técnicas descritas en el capítulo 2 para posteriormente ser conectados por medio de la capa inicial de la red neuronal. Inicialmente se explica cómo las imágenes se convierten en datos y la secuencia de transformaciones a la que es sometida la imagen para luego ser conectada a una red neuronal convencional. Cuando se llega a dicha etapa, la red neuronal busca minimizar el negativo de la log-verosimilitud de la distribución Dirichlet, es decir, ésta es su función de costos. Se minimiza dicha función de costos mediante el ajuste de los parámetros que finalmente servirán para hacer las estimaciones puntuales y por intervalos.

3.1. Modelo

Existen múltiples trabajos que abordan el uso de distribuciones condicionales específicas para ciertas variables de respuesta utilizando las redes neuronales como método de aprendizaje. Por ejemplo, Bishop (1994) realiza un trabajo en el que muestra la estructura de una red neuronal con una distribución condicional para la variable respuesta. El sistema completo es denominado *Mixture Density Networks* en el que se introducen distribuciones condicionales arbitrarias de la misma manera en la que una red neuronal convencional puede representar funciones arbitrarias.

Siguiendo la estructura de las *Mixture Density Networks*, en esta sección se construye un modelo de redes neuronales utilizando la distribución Dirichlet como la distribución condicional de la variable respuesta.

En primer lugar, se asume que se tienen n imágenes con las mismas dimensiones y que contienen las 3 escalas de los colores convencionales RGB. Es decir, cada imagen se puede representar de la siguiente manera para la j-ésima observación (imagen):

$$\boldsymbol{Q^j} = [\boldsymbol{Q_r^j}, \boldsymbol{Q_g^j}, \boldsymbol{Q_b^j}], \ \ \forall \ j = 1, 2, \ldots, n,$$

en donde $\boldsymbol{Q_r^j}$, $\boldsymbol{Q_g^j}$ y $\boldsymbol{Q_b^j}$, son las tres matrices en referencia a los elementos o colores RGB, respectivamente. Asimismo, cada matriz tiene un tamaño de F filas y C columnas. A continuación, se

muestran los elementos de la j-ésima observación en la i-ésima escala de RGB.

$$\boldsymbol{Q_i^j} = \begin{bmatrix} q_{i11}^j & q_{i12}^j & \cdots & q_{i1C}^j \\ q_{i21}^j & q_{i22}^j & \cdots & q_{i2C}^j \\ \vdots & \vdots & \ddots & \vdots \\ q_{iF1}^j & q_{iF2}^j & \cdots & q_{iFC}^i \end{bmatrix} \quad \forall \ i \in (r, g, b).$$

Si se tienen 1000 observaciones y la dimensión de cada imagen es de 700 por 500 pixeles, considerando la estructura RGB se tendrían alrededor de mil millones de datos.

Una vez que se tiene la información de las imágenes en matrices se puede comenzar con la aplicación de la convolución y el $max - pooling$. Normalmente se tienen dos etapas de aplicación del operador convolucional y otras dos para el $max - pooling$ de manera intercalada y empezando por la convolución, sin embargo, se establece una generalización para R_R procesos o etapas convolucionales y M procesos de $max - pooling$.

A continuación, definimos el operador convolucional l-ésimo de la primera etapa convolucional para comenzar a reducir la dimensión de las matrices originales sin perder información o características relevantes asociadas a la imagen:

$$\boldsymbol{\Pi_l^1} = \begin{bmatrix} \pi_{l11}^1 & \pi_{l12}^1 & \cdots & \pi_{l1t}^1 \\ \pi_{l21}^1 & \pi_{l22}^1 & \cdots & \pi_{l2t}^1 \\ \vdots & \vdots & \ddots & \vdots \\ \pi_{ld1}^1 & \pi_{ld2}^1 & \cdots & \pi_{ldt}^1 \end{bmatrix}, \quad \forall \ (d < F, t < C) \in \mathbb{N}, 1 < l \leq R_1.$$

Es importante recalcar que este primer operador convolucional se aplicará a las tres matrices RGB y, como se vio en la Sección 2.4, se sumarán los resultados obtenidos. Asimismo, R_1 corresponde al número de operadores convolucionales aplicados en la primera etapa. Por ejemplo, si la imagen cuenta con 700 por 500 pixeles, y utilizando una dimension de 10×10 para el operador convolucional, es decir $d = 10$ y $t = 10$, además de solo un operador en la primera etapa convolucional: $R_1 = 1$, un $stride = 1$ y un $padding = 0$, se pasaría de tener $1050000 = 700 \times 500 \times 3$ datos a $339281 = (700 - 10 + 1) \times (500 - 10 + 1)$ datos (asumiendo que se usa el mismo operador convolucional para cada una de las tres matrices RGB).

La siguiente ecuación representa todos los operadores convolucionales aplicados en la v-ésima etapa convolucional:

$$\boldsymbol{\Pi^v} = [\boldsymbol{\Pi_1^v}, \boldsymbol{\Pi_2^v}, \ldots, \boldsymbol{\Pi_{R_v}^v}], \quad \forall \ R_v \in \mathbb{N}, 1 < v \leq R, \ \ 1 < R_v \leq R_R,$$

en donde R_R es el número total de etapas convolucionales del modelo. Por otro lado, es fácil notar que no se ha asumido un nivel de $padding$ o $stride$ ya que esto depende del conjunto de datos que

se está analizando y está más ligado a la aplicación del modelo que a su construcción teórica.

Ahora bien, como se indicó anteriormente, la convención del modelo utiliza de manera intercalada el operador convolucional con el $max-pooling$, es así que luego de la aplicación de los primeros R_1 operadores convolucionales en la primera etapa convolucional se aplicará el $max-pooling$ convencional. A continuación, se define la función $max-pooling$ en la g-ésima etapa de $max-pooling$ proveniente de la v-ésima etapa convolucional de determinada d-ésima matriz convolucionada.

$$\sigma_f^{d,g} = max(\boldsymbol{\Sigma_{ae}^{d,v}}), \ \ \forall \ \ g \in \mathbb{N}, 1 < g \leq M.$$

en donde M es el número total de etapas de $max - pooling$, a representa el número de filas del operador $max-pooling$ y e el número de columnas. Asimismo, Σ representa la partición de la matriz generada por el $max - pooling$. Es decir, $\sigma_f^{d,g}$ es el elemento f-ésimo correspondiente a una nueva matriz construída por este proceso de $max-pooling$ que viene de un proceso previo de convolución en su etapa anterior correspondiente a la etapa v-ésima de la d-ésima matriz convolucionada en dicha etapa. Normalmente se debe cumplir que a una etapa de convolución le sigue una etapa de $max - pooling$ y que $R_R = M$.

Las sucesivas aplicaciones intercaladas harán que los elementos originales de las matrices, es decir, los pixeles de la imagen, queden convertidos por las funciones del proceso convolucional t el del $max - pooling$, por lo que al final de los R_R procesos convolucionales y M procesos de $max - pooling$ se tendrán los siguientes valores finales ordenando la matriz final en una de una sola columna, es decir, las columnas de la matriz final son colocadas una debajo de la otra.

$$\boldsymbol{X_j^{(0)}} = \begin{bmatrix} x_{j1} = max^M(\Pi^{R_R}(\dots(\Pi^2(max^1(\Pi^1(q_{i11}^j))))))) \\ x_{j2} = max^M(\Pi^{R_R}(\dots(\Pi^2(max^1(\Pi^1(q_{i21}^j))))))) \\ \vdots \\ x_{jp} = max^M(\Pi^{R_R}(\dots(\Pi^2(max^1(\Pi^1(q_{iF1}^j))))))) \end{bmatrix}, \ \ 1 < j \leq n.$$

El vector de datos $\boldsymbol{X_j^{(0)}}$ está compuesto por p elementos que se ubican en la capa de entrada. Los p elementos corresponden al número de covariables mientras que el indicador j hace referencia a la j-ésima observación o unidad estadística.

Asimismo, cada elemento de la matriz anterior será ajustado por un peso y se le añadirá una constante para poder migrar a una neurona de la siguiente capa. Los pesos y la constante se definen a continuación.

$$\boldsymbol{W_i^{(1)}} = \begin{bmatrix} w_{1i}^{(1)} \\ w_{2i}^{(1)} \\ \vdots \\ w_{pi}^{(1)} \end{bmatrix},$$

$$b_i^{(1)} = \left[b_i^{(1)}\right].$$

El indicador (1) de los vectores anteriores hace referencia a que se está transfiriendo información a la primera capa oculta a partir de la capa de entrada. Por ejemplo, el elemento $w_{1i}^{(1)}$ indica que el valor de la primera covariable multiplicado por este peso se ubicará en la neurona i-ésima de la primera capa oculta. Asimismo, $w_{3i}^{(1)}$ Significa que el valor de la tercera covariable multiplicado por dicho peso se añadirá a la i-ésima neurona de la primera capa oculta. Lo mismo sucede con $b_i^{(1)}$ que significa que dicho valor se añadirá a la neurona i-ésima de la primera capa oculta.

Con estas consideraciones, el valor de la i-ésima neurona de la primera capa se define así:

$$X_{ji}^{(1)} = \sigma(W_i^{(1)^T} X_j^{(0)} + b_i^{(1)}),$$

en donde, $\sigma(x) = max(0, x)$ corresponde a la función sigmoidal mientras que el valor de la i-ésima neurona de la S-ésima capa de la j-ésima observación se define de la siguiente manera.

$$X_{ji}^{(S)} = \sigma(W_i^{(S)^T} X_j^{(S-1)} + b_i^{(S)}).$$

Es decir, el vector de r elementos transformados de $\boldsymbol{X}$ de la j-ésima observación de la S-ésima capa se define de la siguiente manera:

$$\boldsymbol{X_j^{(S)}} = \begin{bmatrix} x_{j1}^{(S)} \\ x_{j2}^{(S)} \\ x_{j3}^{(S)} \\ \vdots \\ x_{jr}^{(S)} \end{bmatrix},$$

en donde cada elemento de la matriz se definen por el valor de los pesos y las neuronas de la capa anterior;

$$\boldsymbol{X_j^{(S)}} = \begin{bmatrix} x_{j1}^{(S)} = \sigma(W_1^{(S)^T} X_j^{(S-1)} + b_1^{(S)}) \\ x_{j2}^{(S)} = \sigma(W_2^{(S)^T} X_j^{(S-1)} + b_2^{(S)}) \\ x_{j3}^{(S)} = \sigma(W_3^{(S)^T} X_j^{(S-1)} + b_3^{(S)}) \\ \vdots \\ x_{jr}^{(S)} = \sigma(W_r^{(S)^T} X_j^{(S-1)} + b_r^{(S)}) \end{bmatrix}.$$

Si se tiene S capas ocultas y cada capa tiene r neuronas además p covariables y K neuronas en

la capa de salida, se tendrán que estimar $r^2 \times (S-1) + p \times r + r \times K$ parámetros. En ese sentido, los parámetros deben cumplir con minimizar una función de costos que se introduce utilizando el negativo del logaritmo de la verosimilitud de la distribución Dirichlet.

Para poder diagramar el vínculo entre la CNN y la distribución Dirichlet se presenta la siguiente ecuación:

$$\hat{\boldsymbol{\alpha}}_{\boldsymbol{j}} = \begin{bmatrix} \hat{\alpha}_{j1} = \sigma(W_1^{(S_F)^T} X_j^{(S_F-1)} + b_1^{(S_F)} \\ \hat{\alpha}_{j2} = \sigma(W_2^{(S_F)^T} X_j^{(S_F-1)} + b_2^{(S_F)} \\ \hat{\alpha}_{j3} = \sigma(W_3^{(S_F)^T} X_j^{(S_F-1)} + b_3^{(S_F)} \\ \vdots \\ \hat{\alpha}_{jK} = \sigma(W_K^{(S_F)^T} X_j^{(S_F-1)} + b_K^{(S_F)} \end{bmatrix},$$

donde S_F es la última capa de la CNN y, como se puede apreciar, el objetivo de la red es estimar los elementos de α_j de cada unidad estadística, para que a partir de allí se puedan estimar los elementos de Y_j por medio de la esperanza. Asimismo, los intervalos de confianza también provendrán de los $\hat{\boldsymbol{\alpha}}_{\boldsymbol{j}}$ estimados, que como se nota, cada unidad estadística posee un vector estimado de parámetros especifico.

Recordemos que se planteó la distribución Dirichlet en el Capítulo 2. Se define de nuevo dicha distribución utilizando los elementos de la variable de respuesta $Y_j = [y_{j1}, y_{j2}, ..., y_{jK}]^T$ que pertenecen a un conjunto de datos composicionales de la j-ésima observación, es decir, la suma de dichos elementos es igual a la unidad y ningún elemento es negativo.

$$f(Y_j; \alpha_j) = f(y_{j1}, y_{j2}, ..., y_{jK}; \alpha_j) = \frac{\Gamma(\sum_{k=1}^K \alpha_{jk})}{\prod_{k=1}^K \Gamma(\alpha_{jk})} \prod_{k=1}^K y_{jk}^{\alpha_{jk}-1}. \tag{3.1}$$

En la ecuación 3.1, $\alpha_j = [\alpha_{j1}, \alpha_{j2}, \ldots, \alpha_{jK}]^T$, asimismo, el indicador j-ésimo se encuentra acotado así: $1 \leq j \leq n$, es decir, funciona para cualquier observación.

Por otro lado, asumiendo la independencia entre las n observaciones se define la función de verosimilitud:

$$\mathcal{L}(\boldsymbol{\alpha}) = \prod_{j=1}^n \left(\frac{\Gamma(\sum_{k=1}^K \alpha_{jk})}{\prod_{k=1}^K \Gamma(\alpha_{jk})} \prod_{k=1}^K y_{jk}^{\alpha_{jk}-1} \right),$$

donde: $\boldsymbol{\alpha} = [\alpha_1^T, \alpha_2^T, \ldots, \alpha_n^T]^T$. Ahora se calcula el logarítmo de la función de verosimilitud para tener la función de log-verosimilitud.

$$\ell(\boldsymbol{\alpha}) = \ln(\mathcal{L}(\boldsymbol{\alpha})) = ln \left(\prod_{j=1}^n \left(\frac{\Gamma(\sum_{k=1}^K \alpha_{jk})}{\prod_{k=1}^K \Gamma(\alpha_{jk})} \prod_{k=1}^K y_{jk}^{\alpha_{jk}-1} \right) \right).$$

Para reducir la ecuación de log-verosimilitud se reduce a partir de la observación j-ésima que permitirá generalizar los resultados:

$$ln(f(Y_j;\alpha_j)) = ln\left(\prod_{k=1}^{K} y_{jk}^{\alpha_{jk}-1}\right) + ln\left(\frac{\Gamma(\sum_{k=1}^{K}\alpha_{jk})}{\prod_{k=1}^{K}\Gamma(\alpha_{jk})}\right).$$

$$ln(f(Y_j;\alpha_j)) = ln\left(\prod_{k=1}^{K} y_{jk}^{\alpha_{jk}-1}\right) + ln\left(\Gamma(\sum_{k=1}^{K}\alpha_{jk})\right) - ln\left(\prod_{k=1}^{K}\Gamma(\alpha_{jk})\right).$$

$$ln(f(Y_j;\alpha_j)) = \sum_{k=1}^{K} ln(y_{jk}^{\alpha_{jk}-1}) + ln\left(\Gamma(\sum_{k=1}^{K}\alpha_{jk})\right) - \sum_{k=1}^{K} ln(\Gamma(\alpha_{jk})).$$

$$ln(f(Y_j;\alpha_j)) = \sum_{k=1}^{K}(\alpha_{jk}-1)ln(y_{jk}) + ln\left(\Gamma(\sum_{k=1}^{K}\alpha_{jk})\right) - \sum_{k=1}^{K} ln(\Gamma(\alpha_{jk})).$$

Por lo tanto, la función de log-verosimilitud se define de la siguiente manera:

$$\ell(\boldsymbol{\alpha}) = \sum_{j=1}^{n}\left(\sum_{k=1}^{K}(\alpha_{jk}-1)ln(y_{jk}) + ln\left(\Gamma(\sum_{k=1}^{K}\alpha_{jk})\right) - \sum_{k=1}^{K} ln(\Gamma(\alpha_{jk}))\right).$$

Como se describió en la Sección 2.3, el modelo de redes neuronales necesita una función de costos que, en este contexto, será el negativo de la función de log-verosimilitud y por lo tanto, la función a minimizar se expresa de la siguiente manera:

$$\mathcal{J}(\boldsymbol{Y},\hat{\boldsymbol{\alpha}}) = -\sum_{j=1}^{n} ln(f(Y_j;\hat{\alpha_j})),$$

$$\mathcal{J}(\boldsymbol{Y},\hat{\boldsymbol{\alpha}}) = -\sum_{j=1}^{n}\left(\sum_{k=1}^{K}(\hat{\alpha}_{jk}-1)ln(y_{jk}) + ln\left(\Gamma(\sum_{k=1}^{K}\hat{\alpha}_{jk})\right) - \sum_{k=1}^{K} ln(\Gamma(\hat{\alpha}_{jk}))\right). \tag{3.2}$$

donde: $\hat{\boldsymbol{\alpha}} = [\hat{\alpha}_1^T, \hat{\alpha}_2^T, \ldots, \hat{\alpha}_n^T]^T$, $\hat{\alpha}_j = [\hat{\alpha}_{j1}, \hat{\alpha}_{j2}, \ldots, \hat{\alpha}_{jK}]^T$, $\boldsymbol{Y} = [\boldsymbol{Y}_1^T, \boldsymbol{Y}_2^T, \ldots, \boldsymbol{Y}_n^T]^T$ y $\boldsymbol{Y}_j = [y_{j1}, y_{j2}, \ldots, y_{jK}]^T$

La ecuación 3.2 es la función de costos global. Es decir, la red neuronal convolucional debe ser capaz de minimizar la función anterior encontrando el vector de α_j óptimo dados los valores del vector composicional por cada unidad estadística. Es importante notar que cuando se minimice la función de costos se estimarán los parámetros α_j por cada unidad estadística por lo que se podrá estimar la distribución Dirichlet para cada observación. Se puede usar la media condicional visto en el Capítulo 2 como predictor para cada elemento del vector composicional de la unidad estadística.

3.2. Estimación del modelo

El método ampliamente utilizado para la estimación de este tipo de modelos es el de $backpropagation$. El método comienza con el planteamiento de la función de costos y las siguientes expresiones:

$$\mathcal{C}(\boldsymbol{Y}, \boldsymbol{\alpha}) = \mathcal{J}(\boldsymbol{Y}, \hat{\boldsymbol{\alpha}}). \tag{3.3}$$

La ecuación 3.3 representa una función de costos general en la que se asocia los valores de salida reales y los valores de salida de la red neuronal convolucional. Asimismo, se define de manera general $z_j^{(L)}$ como el elemento que se consigue en la etapa L-ésima al momento de ponderar los valores de la capa anterior $x_k^{(L-1)}$ y la constante b_j^L. La ecuación 3.4 lo muestra:

$$z_j^{(L)} = \sum_{k=1}^{N_{(L)}} w_{jk}^{(L)} x_k^{(L-1)} + b_j^{(L)}. \tag{3.4}$$

Por último, la ecuación 3.5 muestra el valor del elemento que se obtiene cuando se aplica la función de activación $\sigma(\cdot)$:

$$x_k^{(L)} = \sigma(z_j^{(L)}). \tag{3.5}$$

A partir de allí se calculan las derivadas respecto a los pesos $w_{jk}^{(L)}$ con el objetivo de minimizar la función de costos tal como se observa en la ecuación 3.6:

$$\frac{d\mathcal{C}}{dw_{jk}^{(L)}} = \frac{dz_j^{(L)}}{dw_{jk}^{(L)}} \frac{dx_k^{(L)}}{dz_j^{(L)}} \frac{d\mathcal{C}}{dx_k^{(L)}}. \tag{3.6}$$

Finalmente, se comienza un proceso de iteración dominado bajo la ecuación 3.7:

$$w_{jk}^{(L)}(t+1) = w_{jk}^{(L)}(t) - \eta \frac{d\mathcal{C}}{dw_{jk}^{(L)}}, \tag{3.7}$$

en donde η representa una tasa de aprendizaje.

3.3. Criterios de comparación para la selección de modelos

Dentro de la literatura existente y de acuerdo a Duerr et al. (2020), uno de los criterios considerados para medir el desempeño de un modelo probabilístico de este tipo es utilizar la verosimilitud conjunta de los datos observados, para ser más preciso, utilizar el valor negativo de la log-verosimilitud como medida de desempeño. En este sentido, el cálculo de la ecuación 3.2 para los datos entrenados y testeados nos indica el desempeño del modelo. Mientras más negativo sea el valor calculado del negativo de la log-verosimilitud de un modelo respecto a otro, mejor será su desempeño.

Otro de los criterios clásicos utilizados para poder realizar comparaciones entre distintos modelos es el AIC o Criterio de Información de Akaike estudiado por Bozdogan (1987). La definición del

criterio se presenta de la siguiente manera:

$$\text{AIC} = 2p - 2\ell(\boldsymbol{\theta}),$$

en donde p es el número de parámetros del modelo mientras que $\ell(\boldsymbol{\theta})$ es la función de log verosimilitud asociada al modelo. Sin embargo, como se ha venido presentando, cada unidad estadística testeada tendrá su propio número conjunto de parámetros por lo que, en general, cada observación puede llegar a tener su propio AIC. Es por eso que, inicialmente se propone utillizar un AIC promediado definido de la siguiente manera:

$$\overline{\text{AIC}} = \frac{\sum_{j=1}^{n} \text{AIC}_j}{n},$$

en donde n es el número de datos que se tiene. Asimismo, es preferible calcular este criterio con los datos testeados o de validación.

Finalmente, Hijazi y Jernigan (2009) proponen utilizar como criterio para la selección de un modelo, respecto a otro, el test del ratio de verosimilitud propuesto por Casella y Berger (2002) para el caso de regresiones que utilizan la distribución Dirichlet como la distribución condicional de la variable respuesta. En este test se define el ratio de la siguiente manera:

$$R(X) = \frac{\mathcal{L}(\hat{\boldsymbol{\alpha}}_{\boldsymbol{R}}|X)}{\mathcal{L}(\hat{\boldsymbol{\alpha}}_{\boldsymbol{C}}|X)},$$

en donde, $\mathcal{L}(\hat{\boldsymbol{\alpha}}_{\boldsymbol{R}}|X)$ es la función de verosimilitud evaluada en el parámetro estimado $\hat{\boldsymbol{\alpha}}_{\boldsymbol{R}}$ del modelo restringido dados los datos X, mientras que $\mathcal{L}(\hat{\boldsymbol{\alpha}}_{\boldsymbol{C}}|X)$ es la función de verosimilitud evaluada en el parámetro estimado $\hat{\boldsymbol{\alpha}}_{\boldsymbol{C}}$ del modelo completo dados los datos X. Asintóticamente, y bajo la hipótesis nula y c parámetros, es decir, la diferencia de parámetros entre los modelos. El siguiente estadístico se distribuye como una $\chi^2(c)$.

$$Q(X) = -2lnR(X).$$

Es muy importante notar que, dada la naturaleza del modelo que se está estimando, se tendrán tantos ratios (tests) como datos ya que se tiene un conjunto de parámetros estimados por cada dato composicional por lo que se puede considerar un mejor modelo a aquel que, en la mayoría de los casos, los test le sean beneficiosos. También podría realizarse una ponderación sin embargo habría que fijar un criterio para ponderar más un dato composicional que otro.

Capítulo 4

Estudio de Simulación

En este capítulo se realiza un estudio de simulación con el objetivo de evaluar el desempeño del modelo mediante el uso de la libreria $Tensorflow$ cuyo lenguaje de programación es $Python$. Se estiman los parámetros de cada unidad estadística y luego se hacen estimaciones puntuales por medio de la esperanza condicional. Asimismo, se construyen intervalo de predicción utilizando los percentiles 2.5 y 97.5.

4.1. Consideraciones para la simulación

Para poder desarrollar este estudio de simulación se hace uso de la base de datos denominada $\boldsymbol{MNIST}$ que pertenece a la librería $keras$. Dicha información contiene información de 70000 imágenes correspondientes a los números escritos a mano desde el 0 hasta el 9. Es decir, cada imagen contiene un número en específico. La Figura 4.1 muestra los primeros 35 datos.

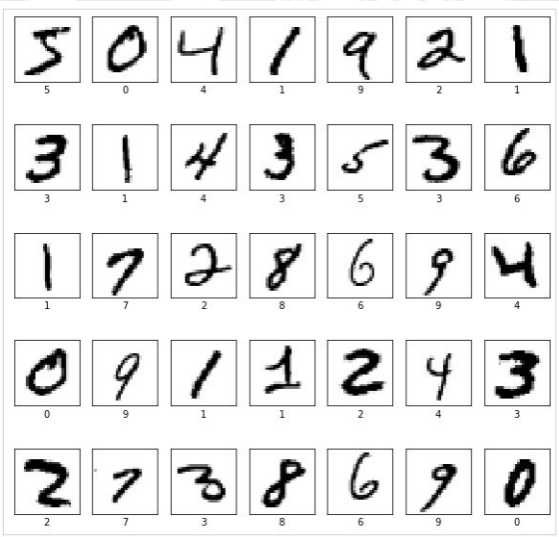

Figura 4.1: Muestra de imágenes originales de todos los números.

Sin embargo, para este estudio solo se trabaja con los números 1, 6, 7 y 8. Esto debido a que, preliminarmente, es más conveniente trabajar con 4 grupos que con 10. Además porque el 1 y el 7 son muy confundidos así como el 6 y el 8 al momento de reconocerlos. La Figura 4.2 presenta un grupo de 35 imágenes asociada a dichos números.

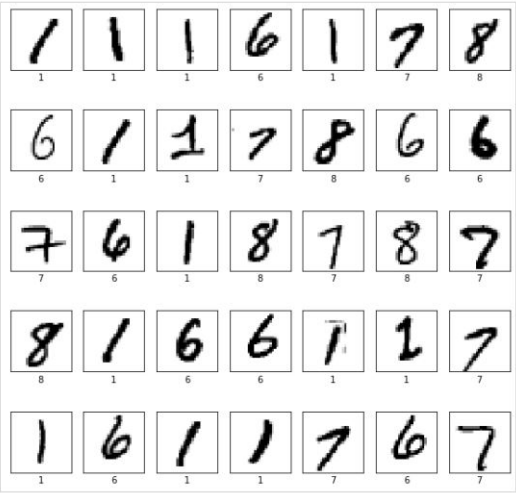

Figura 4.2: Muestra de imágenes de los números 1, 6, 7 y 8.

No es muy difícil notar que el 1 y el 7 son fácilmente confundibles y que el 6 y el 8 presentan una estructura muy similar. Con estas consideraciones el siguiente paso es simular datos composicionales para cada una de estas imágenes.

Si bien se conoce el valor que cada imagen quiere representar, para este estudio simularemos una distribución de probabilidad discreta para cada unidad estadística teniendo en cuenta las consideraciones:

- Se generan cuatro valores de manera aleatoria por cada unidad estadística. Dichos valores aleatorios provienen de distribuciones uniformes con los siguientes parámetros:

$$Z_{j1} \sim U(1, 100), Z_{j2} \sim U(100, 300), Z_{j3} \sim U(300, 500), Z_{j4} \sim U(500, 1000).$$

  En donde el indicador j hace referencia a la j-ésima observación.

- Posteriormente se ponderan los valores simulados, es decir, se divide cada valor entre la suma de todos los valores simulados por unidad estadística de tal manera que se obtienen los datos composicionales.

- Finalmente, se asigna una distribución de probabilidad por cada imagen dandole la mayor probabilidad al valor que realmente pertenece la imagen, la segunda mayor probabilidad se le asigna al par que más se le parezca visualmente. Si la imagen contiene el número 1 entonces la segunda mayor probabilidad va al número 7 y los dos restantes valores se distribuyen aleatoriamente entre el 6 y 8. Lo mismo sucede al asignar probabilidades cuando la imagen contiene un 6, es decir, se le asigna la segunda mayor probabilidad al 8 y las dos restantes se reparten aleatoriamente entre el 1 y el 7. Es decir: $Y = 1, 6, 7, 8$.

Una vez que se tienen los datos simulados se puede observar en el siguiente Cuadro 4.1 la distribución de probabilidades de los primeros 18 datos con su respectiva distribución de probabilidades

discreta (datos composicionales).

Y	$P(Y=1)$	$P(Y=6)$	$P(Y=7)$	$P(Y=8)$
7	0.234	0.044	0.577	0.145
1	0.560	0.041	0.295	0.104
1	0.510	0.221	0.243	0.027
6	0.089	0.481	0.134	0.297
1	0.520	0.166	0.277	0.037
7	0.372	0.080	0.542	0.006
6	0.070	0.460	0.123	0.348
6	0.172	0.571	0.010	0.246
7	0.254	0.179	0.556	0.012
1	0.575	0.041	0.218	0.165
1	0.537	0.174	0.281	0.008
7	0.261	0.187	0.527	0.026
7	0.235	0.103	0.617	0.045
1	0.443	0.059	0.304	0.195
1	0.475	0.059	0.310	0.156
1	0.636	0.090	0.233	0.041
7	0.335	0.011	0.501	0.153
1	0.644	0.085	0.221	0.05
...	...	...	...	...

Cuadro 4.1: Distribución de probabilidades de los datos de test.

Ya se ha notado que una distribución de probabilidades discreta es un dato composicional por lo que en adelante se usarán ambos terminos de manera intercambiable. Finalmente, el objetivo de la red neuronal convolucional será predecir los parámetros que puedan describir una distribución por cada unidad estadística y se hará la estimación de las probabilidades por medio de la esperanza condicional.

La Figura 4.2 nos muestra la estructura de la red neuronal convolucional del modelo especificado para este modelo. El modelo CNN se entrena con 24776 imágenes y se hace una prueba de predicción o test con 4095 observaciones.

Action	**Activation Shape**	**Activation Size**	**Num Parameters**
Input:	(28,28,1)	784	0
Convolution 1:	(26,26,32)	21632	320
Max pooling 1:	(13,13,32)	5408	0
Convolution 2:	(11,11,64)	7744	18496
Max pooling 2:	(5,5,64)	1600	0
Flatten:	(1600,1)	1600	0
Dense:	(4,1)	4	6404

Cuadro 4.2: Parámetros del modelo para la simulación.

En primer lugar, el tamaño de cada imagen es de 28 pixeles de altura por 28 pixeles de anchura y solo se tiene una capa, es decir, no tiene el formato RGB por lo que el tamaño de cada imagen es de (28,28,1), es decir, cada imagen contiene 784 datos numéricos. La primera convolución es del tamaño

(3,3) y se utilizan 32 operadores convolucionales, es por ello que la dimensión del ancho y alto pasa de 28 a 26=28-3+1. Y de tener 1 sola capa a 32 por lo que el tamaño de los datos de activación es de (26,26,32) y se generan los primeros 320=(3*3*1+1)*32 parámetros iniciales. Posteriormente, el $max - pooling$ es del tamaño (2,2) es por ello que se pasa de tener 26 valores a 13, en esta parte del proceso el tamaño de los datos de activación se queda en (13,13,32). El siguiente paso es cuando se aplica el segundo conjunto de operadores convolucionales. De nuevo, se utiliza el mismo tamaño del operador convolucional de la primera etapa sin embargo, esta vez se aplican 64 operadores convolucionales por lo que se obtiene la siguiente estructura de datos de activación: (11,11,64) Asimismo, se generan otros 18496=(3*3*32+1)*64 nuevos parámetros. El siguiente proceso corresponde al $max - pooling$, el operador es de tamaño (2,2) descartandose los bordes es por ello que se pasa de un tamaño 11 a 5, un poco menos que la mitad. En esta etapa el tamaño de los datos de activación es de 1600=5*5*64 y se conectan con la red neuronal convencional que tiene un tamaño de salida de (4,1), es decir, se obtendrán 4 valores por cada imagen que ingrese a la red. Valores que habrán sido minimizados con la función de distribución Dirichlet. Finalmente, es importante recalcar que la función de activación que se utilizó fue la denominada $ReLu$ que tiene la siguiente forma funcional: $f(x) = max(0, x)$.

4.2. Resultados

Los parámetros de los resultados obtenidos se resumen en el Cuadro 4.3 para algunas observaciones pertenecientes a los datos de testeo mas no a los de entrenamiento. Asimismo, se puede notar que el índice j hace referencia a la observación, Y al valor real que representa la imagen y, por ejemplo, la columna $\hat{\boldsymbol{\alpha}}_{\boldsymbol{j6}}$ hace referencia a los parámetros por cada observación asociados al número 6. Es fácil notar que el valor del alfa más elevado por cada observación corresponde al valor del verdadero.

$\boldsymbol{j}$	$\boldsymbol{Y}$	$\hat{\boldsymbol{\alpha}}_{\boldsymbol{j1}}$	$\hat{\boldsymbol{\alpha}}_{\boldsymbol{j6}}$	$\hat{\boldsymbol{\alpha}}_{\boldsymbol{j7}}$	$\hat{\boldsymbol{\alpha}}_{\boldsymbol{j8}}$
1	7	5.938	1.915	11.55	1.587
2	1	7.669	1.461	4.991	1.298
3	1	9.024	1.559	5.856	1.471
4	6	1.624	10.92	1.339	8.496
5	1	8.518	1.444	5.215	1.504
6	7	5.681	1.735	11.79	1.461
7	6	1.780	11.25	1.644	7.643
8	6	1.647	7.542	1.262	4.917
9	7	6.357	1.603	11.04	1.272
...	...	...	...	...	...

Cuadro 4.3: Parámetros estimados del conjunto de datos de test.

Por otro lado, para poder tener la distribución de cada observación se utilizará la esperanza condicional como estimador puntual utilizando las propiedades de la distribución de Dirichlet presentadas en la Sección 2.2. Por lo que, aplicando la propiedad de la esperanza por cada elemento de los datos composicionales se obtiene el Cuadro 4.4 en donde ya se puede observar que se obtiene una distribución por cada observación.

Es muy sencillo notar que en el Cuadro 4.4 la esperanza condicional o probabilidad es la más grande cuando se trata del número que se quiso predecir lo que se alinea a las distribuciones inicialmente generadas en la sección anterior.

j	Y	$E[\hat{Y}_j = 1]$	$E[\hat{Y}_j = 6]$	$E[\hat{Y}_j = 7]$	$E[\hat{Y}_j = 8]$
1	7	0.283	0.091	0.550	0.076
2	1	0.497	0.095	0.324	0.084
3	1	0.504	0.087	0.327	0.082
4	6	0.073	0.488	0.060	0.38
5	1	0.511	0.087	0.313	0.09
6	7	0.275	0.084	0.571	0.071
7	6	0.080	0.504	0.074	0.342
8	6	0.107	0.491	0.082	0.32
9	7	0.313	0.079	0.545	0.063
...	...	...	...	...	...

Cuadro 4.4: Esperanzas condicionales.

Si se usa como predicción al valor que tenga mayor probabilidad se obtiene que se pueden pronosticar correctamente el 98.6 % de los datos de testeo de manera correcta. Por último, en este contexto, se pueden generar intervalos de predicción para cada probabilidad a partir de 1000 valores simulados con los parámetros estimados de cada dato composicional, aplicar la propiedad de la esperanza y posteriormente tomar los percentiles 2.5 y 97.5. El Cuadro 4.5 muestra dichos intervalos de predicción.

j	Y	$Q_{2,5}^{1,j} - Q_{97,5}^{1,j}$	$Q_{2,5}^{6,j} - Q_{97,5}^{6,j}$	$Q_{2,5}^{7,j} - Q_{97,5}^{7,j}]$	$Q_{2,5}^{8,j} - Q_{97,5}^{8,j}$
1	7	0.116 - 0.496	0.011 - 0.237	0.334 - 0.753	0.007 - 0.218
2	1	0.249 - 0.728	0.007 - 0.271	0.118 - 0.569	0.005 - 0.276
3	1	0.279 - 0.716	0.008 - 0.277	0.134 - 0.568	0.006 - 0.229
4	6	0.006 - 0.210	0.281 - 0.694	0.004 - 0.182	0.187 - 0.588
5	1	0.270 - 0.735	0.006 - 0.263	0.125 - 0.550	0.005 - 0.258
6	7	0.120 - 0.472	0.009 - 0.250	0.353 - 0.773	0.005 - 0.208
7	6	0.009 - 0.214	0.307 - 0.699	0.008 - 0.203	0.167 - 0.554
8	6	0.011 - 0.305	0.236 - 0.737	0.004 - 0.257	0.121 - 0.546
9	7	0.136 - 0.514	0.007 - 0.230	0.345 - 0.752	0.004 - 0.195
...	...	...	...	...	...

Cuadro 4.5: Intervalos de predicción.

Para la construcción de cada intervalo de confianza se tomaron los percentiles 2.5 y 95 para los límites inferior y superior, respectivamente. La manera de calcularlos fue generando 1000 valores aleatorios en el programa R utilizando los parámetros que se predijeron por medio de la red neuronal convolucional.

Capítulo 5

Aplicación

En este capítulo se presenta una aplicación real del modelo que se ha planteado anteriormente. El contexto en el que se entrena este modelo corresponde a la industria textil de la moda. En concreto, se busca predecir las participaciones de tallas (datos composicionales) por medio de la imagen de la prenda.

5.1. Descripción de la base de datos

En primer lugar, es importante señalar el origen y condiciones de la base de datos. En esa medida, la información proviene de una compañía de venta directa de cierto país latinoamericano cuyo nombre se mantiene en reserva. Dicha compañía vende por catálogo cuya renovación es de 21 días, es decir, cada 21 días el portafolio de productos vendidos cambia. Dicho esto, es fácil inferir que se tienen alrededor de 18 catálogos por año. Se toman datos desde el año 2017 hasta, de manera parcial, el 2021 para los grupos más extremos de consumidores (prendas con tendencia a vender tallas muy grandes y prendas con tendencia a vender tallas muy pequeñas).

Como se tiene una amplia variedad de prendas de vestir como blusas, vestidos, pantalones, chaquetas, etc, para este caso se ha elegido trabajar con blusas que hayan sido lanzados al mercado en las tallas $Small$, $Medium$, $Large$ y $ExtraLarge$. Con este rango temporal y una vez escogidas las blusas como las prendas a modelar se logran obtener alrededor de 116 observaciones. Cada blusa tiene una nombre y participaciones de tallas. A continuación, se hace una descripción de la base de datos:

- INDICADOR: Es un indicador simple de conteo.
- T.S: Indica la participación de la venta de la blusa en la talla $Small$.
- T.M: Indica la participación de la venta de la blusa en la talla $Medium$.
- T.L: Indica la participación de la venta de la blusa en la talla $Large$.
- T.XL: Indica la participación de la venta de la blusa en la talla $ExtraLarge$.

La Figura 5.1 nos presenta las fotos originales de una muestra de 9 blusas.

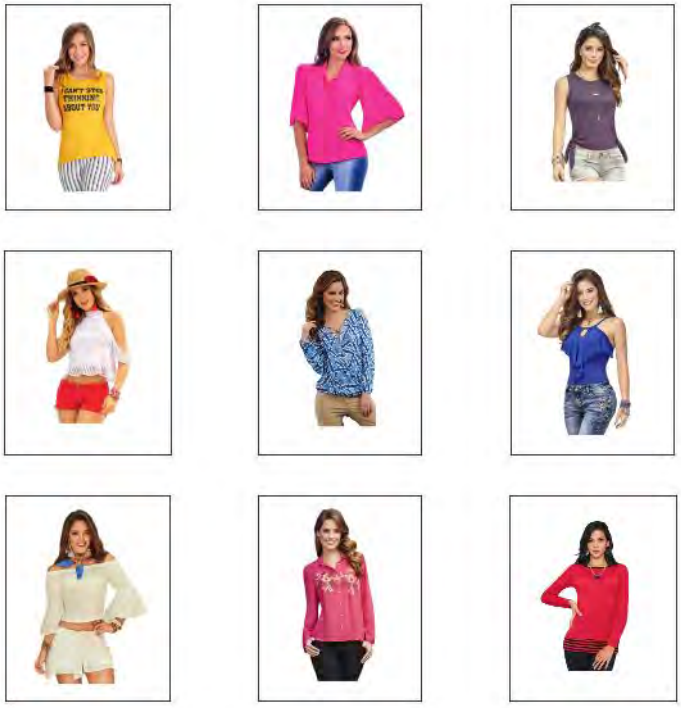

Figura 5.1: Imágenes originales de las blusas.

Se puede notar que las blusas han sido presentadas en un fondo blanco y puestos a la misma altura. Adicionalmente, en el Cuadro 5.1 se muestran los datos composicionales asociados a las nueve observaciones.

IND	$T.S$	$T.M$	$T.L$	$T.XL$
1	0.464	0.289	0.156	0.090
2	0.108	0.272	0.268	0.352
3	0.510	0.315	0.128	0.046
4	0.640	0.252	0.077	0.030
5	0.100	0.286	0.307	0.306
6	0.479	0.352	0.102	0.067
7	0.528	0.284	0.137	0.050
8	0.139	0.269	0.303	0.288
9	0.114	0.282	0.341	0.263

Cuadro 5.1: Participación de venta según la talla.

Por otro lado, las imágenes han sido tratadas para poder ser consideradas como inputs para el modelo, se han replicado los colores extensión y posición de todas las blusas respetando su color RGB así como se ha mantenido una estandarización en ubicación dentro de un fondo blanco para que el modelo pueda aprender mejor. La Figura 5.2 lo muestra para las 9 imágenes muestreadas.

Adicionalmente, la Figura 5.3 muestra el comportamiento de las participaciones de las tallas por cada dato composicional para todas las observaciones divididas en dos grupos construidos mediante el método de $K - means$.

Figura 5.2: Imágenes Tratadas de las Blusas.

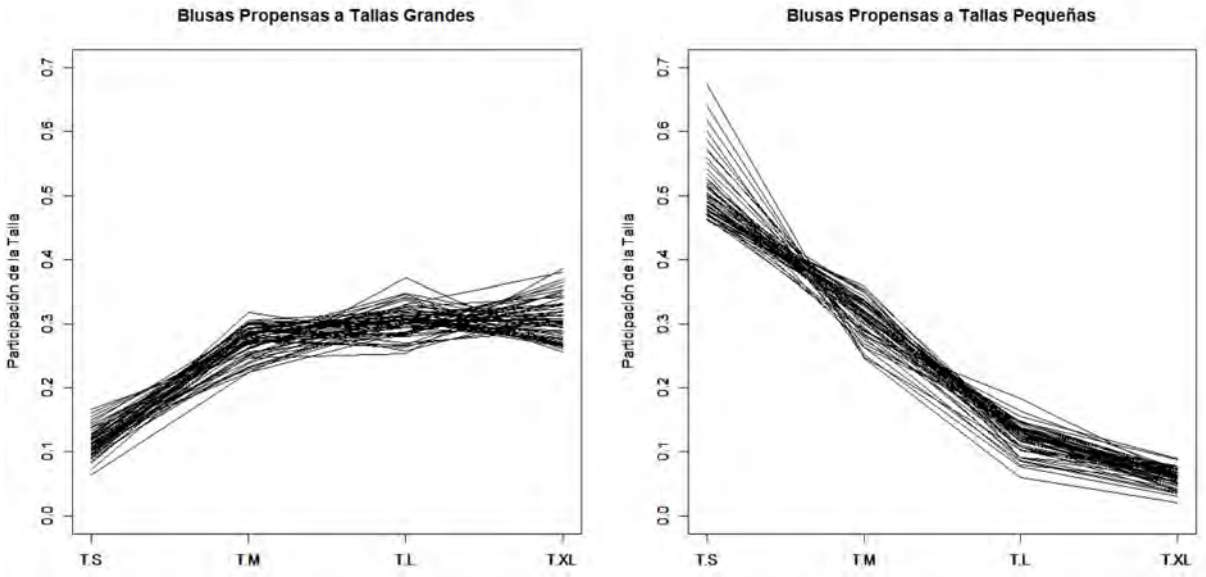

Figura 5.3: Participación de tallas en 2 grupos.

Se puede notar que en el primer grupo se presentan líneas crecientes a medida que la talla es más grande mientras que en el segundo grupo la participación mayor se concentra en las tallas más pequeñas. La interpretación de estos dos grupos o familias de tallas es que, naturalmente, hay prendas diseñadas más apropiadamente para personas de mayor contextura mientras que otras para personas de menor contextura.

Originalmente, la dimensión de cada imagen es de 669 $\times$ 842 $\times$ 3. Sin embargo, luego de haber tratado la imagen para estandarizarla como se muestra en la Figura 5.2 el tamaño cambió a uno de 1151 $\times$ 151 $\times$ 3. Posteriormente se ha hecho una separación entre datos de entrenamiento y datos de testeo. Se han considerado 95 imágenes para entrenar el modelo y 21 para testearlo. Asimismo, ya que solo se cuentan con pocos datos, se hizo un proceso de aumentación de datos tal como lo recomienda Krizhevsky et al. (2017). Concretamente, se utilizaron el método del rotamiento para aumentar la candidad de datos de entrenamiento para posteriormente reducir su tamaño a 28 $\times$ 28 $\times$ 3. La Figura 5.4 muestra un ejemplo para cierta blusa.

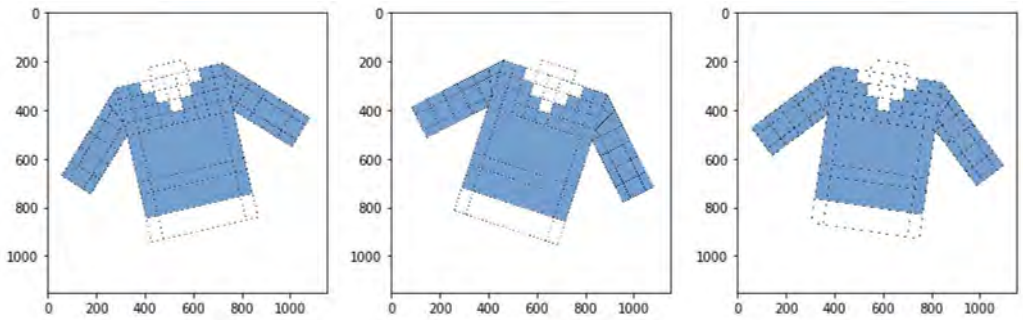

Figura 5.4: Aumentación de datos por rotación.

Con este proceso se consiguieron de 6650 datos ya que por cada foto de entrenamiento se replicaron 70 utilizando la distribución uniforme como generador aleatorio de los grados de rotación entre -20 y 20 grados.

5.2. Modelo final

Originalmente se trabaja con 95 imágenes originales para el entrenamiento pero considerando los aumentados se tienen 6650 observaciones. Con este número de datos se construye el modelo. Se utilizan 21 datos para realizar la comprobación del modelo. Es importante recalcar que los 95 datos originales son de los años 2017 a 2020 mientras que los 21 datos para la realización del test son del año 2021. El Cuadro 5.2 muestra la estructura de la red neuronal:

Action	Activation Shape	Activation Size	Num Parameters
Input:	(28,28,1)	784	0
Convolution 1:	(58,24,32)	44544	320
Max pooling 1:	(29,12,32)	11136	0
Convolution 2:	(27,10,64)	17280	18496
Max pooling 2:	(13,5,64)	4160	0
Flatten:	(4160,1)	4160	0
Dense:	(4,1)	4	16644

Cuadro 5.2: Parámetros del modelo para la aplicación.

En primer lugar, el tamaño de cada imagen es de 28 pixeles de altura por 28 pixeles de anchura y solo se tiene una capa RGB por lo que el tamaño de cada imagen es de (28,28,1), es decir, cada imagen contiene 784 datos numéricos. La primera convolución es del tamaño (3,3) y se utilizan 32 operadores

convolucionales, es por ello que la dimensión del alto pasa de 28 a 26=28-3+1 y el ancho de 28 a 26=28-3+1. Y de tener 1 sola capa a 32 por lo que el tamaño de los datos de activación es de (26,26,32) y se generan los primeros 320=(3*3*1+1)*32 parámetros iniciales. Posteriormente, el $max - pooling$ es del tamaño (2,2) es por ello que se pasa de tener 26 valores a 13, en esta parte del proceso el tamaño de los datos de activación se queda en (13,13,32). El siguiente paso es cuando se aplica el segundo conjunto de operadores convolucionales. De nuevo, se utiliza el mismo tamaño del operador convolucional de la primera etapa sin embargo, esta vez se aplican 64 operadores convolucionales por lo que se obtiene la siguiente estructura de datos de activación: (11,11,64) Asimismo, se generan otros 18496=(3*3*32+1)*64 nuevos parámetros. El siguiente proceso corresponde al $max - pooling$, el operador es de tamaño (2,2) descartandose los bordes es por ello que se pasa de un tamaño 11 a 5, un poco menos que la mitad en el primer caso. En esta etapa el tamaño de los datos de activación es de 1600=5*5*64 y se conectan con la red neuronal convencional que tiene un tamaño de salida de (4,1), es decir, se obtendrán 4 valores por cada imagen que ingrese a la red. Valores que habrán sido minimizados con la función de distribución Dirichlet. En la última parte se obtienen 6404 = 1600*4+4 parámetros. Finalmente, es importante recalcar que la función de activación que se utilizó fue la denominada $ReLu$ que tiene la siguiente forma funcional: $f(x) = max(0, x)$.

5.3. Resultados

Considerando el modelo anterior se obtiene los resultados de los parámetros estimados para los datos de test en el Cuadro 5.3 para las primeras 9 observaciones.

j	$\hat{\alpha}_{jS}$	$\hat{\alpha}_{jM}$	$\hat{\alpha}_{jL}$	$\hat{\alpha}_{jXL}$
1	4.288	6.918	6.069	5.548
2	3.147	7.186	7.843	7.611
3	3.827	6.344	5.792	5.367
4	5.215	4.096	2.23	1.43
5	5.215	4.096	2.23	1.43
6	3.67	5.807	5.538	5.008
7	3.081	7.723	8.359	8.236
8	5.23	4.363	2.426	1.644
9	3.555	7.188	7.293	6.803

Cuadro 5.3: Parámetros estimados para el conjunto de datos de test.

Asimismo, se calculan las estimaciones puntuales por medio de la esperanza en el Cuadro 5.4. Por otro lado, el Cuadro 5.5 muestra los valores reales. Se puede notar que existen diferencias significativas en algunos casos. Finalmente, en el Cuadro 5.6 se construyen los intervalos de confianza utilizando los percentiles 2.5 y 97.5 generados a partir de 1000 valores aleatorios simulados con los parámetros de cada observación testeada.

Por otro lado, la Figura 5.5 muestra los intervalos de predicción de las 4 tallas, es decir, los percentiles 2.5 y 97.5 así como los valores reales y las esperanzas predichas.

No es dificil notar que para este primer dato composicional, el valor predicho más cercano fue para la talla M mientras que el dato más distante para la talla S.

j	$E[\hat{S}_j]$	$E[\hat{M}_j]$	$E[\hat{L}_j]$	$E[\hat{XL}_j]$
1	0.188	0.303	0.266	0.243
2	0.122	0.279	0.304	0.295
3	0.179	0.297	0.272	0.252
4	0.402	0.316	0.172	0.11
5	0.402	0.316	0.172	0.11
6	0.183	0.29	0.277	0.25
7	0.112	0.282	0.305	0.301
8	0.383	0.319	0.178	0.12
9	0.143	0.289	0.294	0.274

Cuadro 5.4: Esperanzas predichas de los datos de test.

j	S_j	M_j	L_j	XL_j
1	0.102	0.263	0.33	0.304
2	0.094	0.274	0.283	0.349
3	0.117	0.303	0.31	0.27
4	0.106	0.278	0.297	0.319
5	0.091	0.283	0.314	0.312
6	0.09	0.249	0.302	0.359
7	0.1	0.289	0.302	0.309
8	0.558	0.283	0.108	0.051
9	0.105	0.305	0.319	0.271

Cuadro 5.5: Valores reales de los datos de test.

j	$Q_{2,5}^{S,j} - Q_{97,5}^{S,j}$	$Q_{2,5}^{M,j} - Q_{97,5}^{M,j}$	$Q_{2,5}^{L,j} - Q_{97,5}^{L,j}$	$Q_{2,5}^{XL,j} - Q_{97,5}^{XL,j}$
1	0.060 - 0.370	0.138 - 0.495	0.104 - 0.452	0.084 - 0.419
2	0.029 - 0.275	0.122 - 0.466	0.143 - 0.502	0.129 - 0.484
3	0.053 - 0.360	0.115 - 0.508	0.100 - 0.472	0.096 - 0.441
4	0.151 - 0.673	0.107 - 0.569	0.030 - 0.408	0.006 - 0.298
5	0.158 - 0.657	0.112 - 0.583	0.027 - 0.415	0.009 - 0.339
6	0.052 - 0.367	0.130 - 0.493	0.104 - 0.478	0.101 - 0.46
7	0.023 - 0.252	0.129 - 0.457	0.155 - 0.493	0.158 - 0.49
8	0.158 - 0.635	0.108 - 0.563	0.030 - 0.388	0.012 - 0.328
9	0.042 - 0.309	0.131 - 0.469	0.139 - 0.489	0.123 - 0.45

Cuadro 5.6: Intervalos de predicción de los datos de test.

Finalmente, las Figuras 5.6 y 5.7 muestran los intervalos de confianza para el resto de datos composicionales testeados. Se puede notar que en los datos 4, 5, 11, 19 y 20 ciertos componentes de los datos composicionales testeados se encuentran lejos de los valores reales y tienen a acercarse mucho o sobrepasar el límite superior definido. En algunos casos inclusive la tendencia testeada es contraria a la tendencia real. Es decir, se predice una curva de tallas como si la prenda fuera a venderse más en tallas pequeñas pero realmente se vende más en tallas grandes y viceversa.

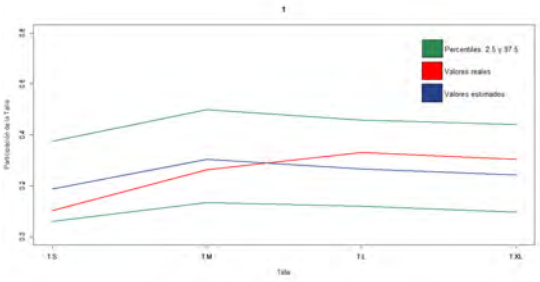

Figura 5.5: Intervalos de predicción para el primer dato de test.

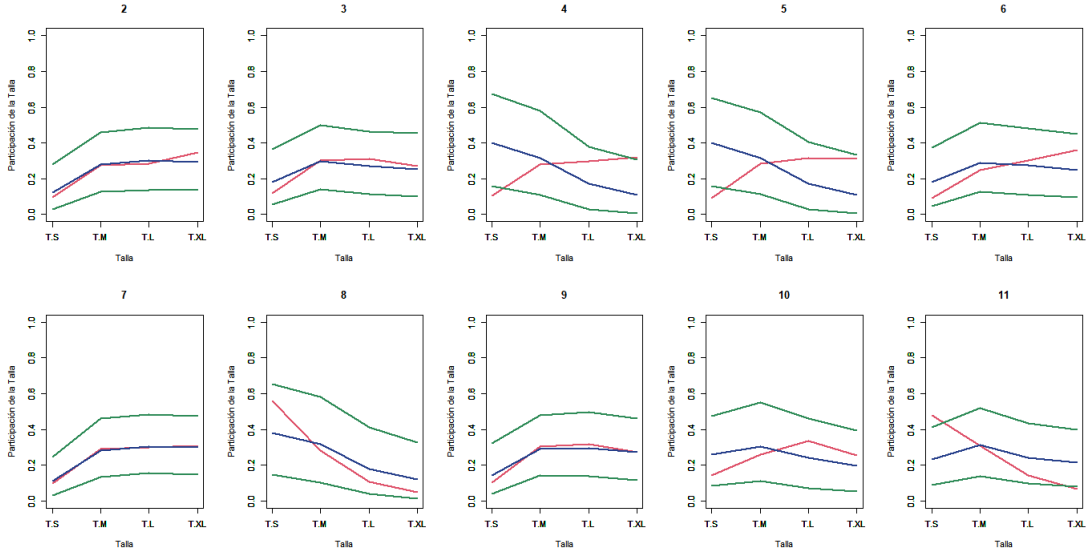

Figura 5.6: Intervalos de predicción de los datos de test (2-11).

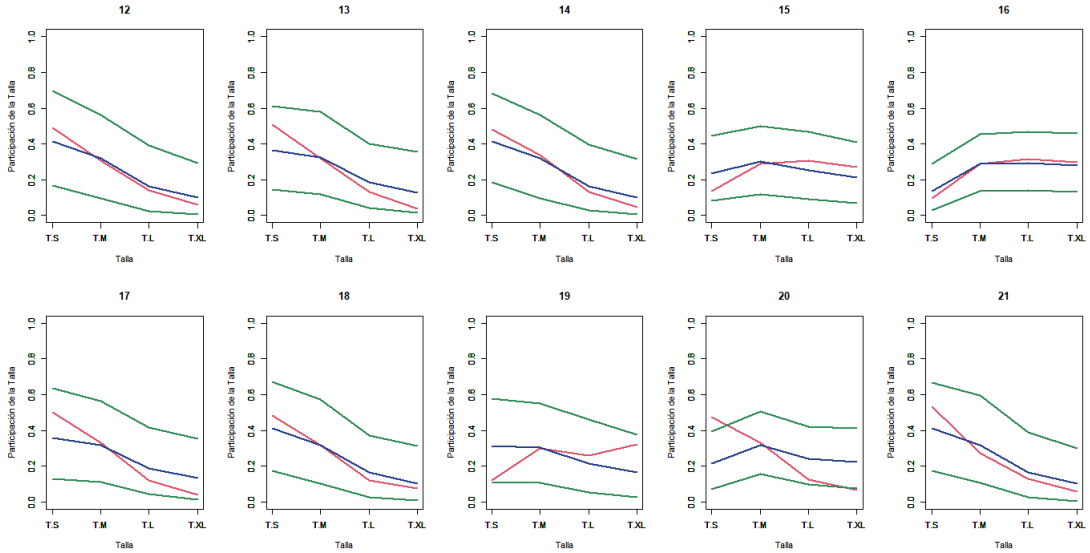

Figura 5.7: Intervalos de predicción de los datos de test (12-21).

Capítulo 6

Conclusiones y Sugerencias

6.1. Conclusiones

En este trabajo de tesis se ha desarrollado un modelo de Redes Neuronales Convolucionales para Datos Composicionales utilizando el negativo de la log-verosimilitud de la función de Distribución de Dirichlet como función de pérdida. Uno de los principales beneficios de utilizar una función de pérdida asociado a una distribución es que se podrán generar estadísticas de predicción más diversas y completas a las que convencionalmente arrojan los modelos CNN tradicionales que utilizan funciones de pérdida como el error cuadrático medio para ejercicios de regresión o la entropía para problemas de clasificación. El diseño de este tipo de modelo permite construir intervalos de confianza o poder hacer predicciones no solo con la esperanza o promedio sino con los percentiles ya que se obtiene distintos parámetros por cada unidad estadística.

En el estudio de simulación se lograron resultados bastante buenos llegándose a pronosticar alrededor del 98.6 % de imágenes de manera correcta por medio de la elección de la esperanza más grande como estimación puntual del número en análisis. Asimismo, es importante recalcar que se contaba con alrededor de 25 mil datos (imágenes) para entrenar y cerca de 4 mil datos para testear.

Por otro lado, en el estudio de aplicación las esperanzas para los datos testeados muestra que en su mayoría, casi el 76 % de los datos predichos presenta una tendencia razonable en cuanto al tipo de prenda (más porcentaje predicho en tallas grandes cuando realmente se han vendido tallas grandes y viceversa). Mientras que para el 24 % de los datos la tendencia testeada es contraria, más fuerte o más débil respecto de los datos reales.

Finalmente, la calidad de la predicción puede mejorar notablemente en la medida que se incrementen más imágenes de entrenamiento. Asimismo, previamente se hicieron pruebas con fotos originales para vestidos pero los resultados obtenidos no fueron los esperados ya que la imagen original, en muchos casos, no muestra la información del tamaño real de la prenda como para que el modelo pueda distinguir una curva de talla más apropiada.

6.2. Sugerencias para investigaciones futuras

- Asociar el modelo a una evolución temporal. En este trabajo se tratan los datos como si provenieran de una estructura de corte transversal, es decir, no hay un parámetro que refleje

temporalidad, sin embargo puede ser de mucha utilidad incluirlo ya que con el tiempo es razonable pensar que la población que compra las prendas va cambiando su estructura fisiológica por lo que su talla va mutando y en consecuencia sus compras hacia prendas más o menos grandes.

- Implementar un modelo bayesiano para la aplicación. Debido a que existen condicionantes que inicialmente pueden sentar una primera piedra acerca del comportamiento de la distribución de las tallas como el color o el tipo de material, se pueden asignar distribuciones a priori para un mejor pronóstico.
- Probar otras funciones de distribución del tipo mixtura que sean adecuadas para datos composicionales.

Apéndice A

Rutinas en Python

A.1. Rutinas en Python para la Simulación

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers

num_classes = 4
input_shape = (28, 28, 1)

(x_train, y_train), (x_test, y_test) = keras.datasets.
mnist.load_data()

### Ahora solo nos quedamos con los valores que sean iguales a
# 1, 6, 8 y 7:

### Primero para los datos de entrenamiento:

y_train_1 = y_train[(y_train==1) | (y_train==7) | (y_train==6)
 | (y_train==8)]
x_train_1 = x_train[(y_train==1) | (y_train==7) | (y_train==6)
| (y_train==8)]

### Luego para los datos de validación:

y_test_1 = y_test[(y_test==1) | (y_test==7) | (y_test==6)
 | (y_test==8)]
x_test_1 = x_test[(y_test==1) | (y_test==7) | (y_test==6)
 | (y_test==8)]

# Scale images to the [0, 1] range
```

```
x_train = x_train_1.astype("float32") / 255

x_test = x_test_1.astype("float32") / 255
# Make sure images have shape (28, 28, 1)

prueba = np.expand_dims(x_train[1],-1)

x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")

############################################################
############################################################

import pandas as pd

y_test_new = pd.read_excel(r'C:\Users\PAVEL\Dropbox\
3.- TESIS - PAVEL COTACALLAPA\4.- DESARROLLO DE LA TESIS\
3.- DATOS\3.- VALORES\y_test_out_1_ajus.xlsx')
y_train_new = pd.read_excel(r'C:\Users\PAVEL\Dropbox\
3.- TESIS - PAVEL COTACALLAPA\4.- DESARROLLO DE LA TESIS\
3.- DATOS\3.- VALORES\y_train_out_1_ajus.xlsx')

y_test_new.head()
y_train_new.head()

### Ahora le damos la forma de

y_train_new_1 = np.array(y_train_new, dtype="float32")
y_test_new_1 = np.array(y_test_new, dtype="float32")

# convert class vectors to binary class matrices

y_train = y_train_new_1
y_test = y_test_new_1

model = keras.Sequential(
[
keras.Input(shape=input_shape),
```

```
layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
layers.MaxPooling2D(pool_size=(2, 2)),
layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
layers.MaxPooling2D(pool_size=(2, 2)),
layers.Flatten(),
layers.Dropout(0.5),
#    layers.Dense(num_classes, activation="softmax"),
layers.Dense(num_classes, activation="relu"),
]
)

model.summary()

batch_size = 128
epochs = 9

######################################################
######################################################

### Escribimos la función de pérdida explicitamente:

import tensorflow as tf
import math

def my_loss(y_true,y_pred):

alfa_1 = tf.slice(y_pred,[0,0],[-1,1])
alfa_2 = tf.slice(y_pred,[0,1],[-1,1])
alfa_3 = tf.slice(y_pred,[0,2],[-1,1])
alfa_4 = tf.slice(y_pred,[0,3],[-1,1])

y_1 = tf.slice(y_true,[0,0],[-1,1])
y_2 = tf.slice(y_true,[0,1],[-1,1])
y_3 = tf.slice(y_true,[0,2],[-1,1])
y_4 = tf.slice(y_true,[0,3],[-1,1])

y_pot_1 = y_1**(alfa_1-1)
y_pot_2 = y_2**(alfa_2-1)
y_pot_3 = y_3**(alfa_3-1)
y_pot_4 = y_4**(alfa_4-1)

epsilon = tf.constant([0.0001])
```

```
gamma_1=tf.math.exp(tf.math.lgamma(tf.math.abs(alfa_1+epsilon)))
gamma_2=tf.math.exp(tf.math.lgamma(tf.math.abs(alfa_2+epsilon)))
gamma_3=tf.math.exp(tf.math.lgamma(tf.math.abs(alfa_3+epsilon)))
gamma_4=tf.math.exp(tf.math.lgamma(tf.math.abs(alfa_4+epsilon)))

#gamma_1 = tf.math.exp(tf.math.lgamma(alfa_1))
#gamma_2 = tf.math.exp(tf.math.lgamma(alfa_2))
#gamma_3 = tf.math.exp(tf.math.lgamma(alfa_3))
#gamma_4 = tf.math.exp(tf.math.lgamma(alfa_4))

gamma_all = tf.math.exp(tf.math.lgamma(alfa_1+alfa_2+alfa_3+
alfa_4+epsilon))

up = y_pot_1*y_pot_2*y_pot_3*y_pot_4

arthur = (gamma_all*up)/(gamma_1*gamma_2*gamma_3*gamma_4)

arthur_1 = -tf.math.log(arthur)

loss = arthur_1
# loss = tf.reduce_sum(-tf.math.log(arthur),axis=0)

return loss

############################################################
############################################################

### Compilamos el modelo::

model.compile(loss=my_loss, optimizer="adam", metrics=[my_loss])
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
 validation_split=0.1)

############################################################
############################################################

### En esta sección calculamos la función de pérdida de
 manera manual ya que se exportan
### libros de excel:

test_loss,test_acc=model.evaluate(x_test,  y_test, verbose=2)
```

```
train_loss,train_acc=model.evaluate(x_train, y_train,verbose=2)

predictions_test = model.predict(x_test)

import pandas as pd

predictions_test_1 = pd.DataFrame(predictions_test)

sumita = predictions_test_1.sum(axis=1)

############################################################
############################################################

### Exportamos los valores de testeo:

import pandas as pd

Datos_testeados = predictions_test_1

Datos_testeados.to_excel(r'C:\Users\PAVEL\Dropbox\3.- TESIS -
PAVEL COTACALLAPA\4.- DESARROLLO DE LA TESIS\4.- MODELOS\1.-
  SIMULACION\Datos_testeados.xlsx', index = False)

### Ahora se exportan los valores reales:

Datos_reales = pd.DataFrame(y_test)

Datos_reales.to_excel(r'C:\Users\PAVEL\Dropbox\3.- TESIS - PAVEL
 COTACALLAPA\4.- DESARROLLO DE LA TESIS\4.- MODELOS\1.-
  SIMULACION\Datos_reales.xlsx', index = False)
```

A.2. Rutinas en Python para la Aplicación

```
### En este código se elabora un modelo para predecir las curvas
de tallas:

### En primer lugar se carga la base de datos con los valores
y nombres de cada prenda:

import pandas as pd
from PIL import Image
```

```
import numpy as np

datos_train = pd.read_excel('C:/Users/PAVEL/Documents/
MIS DOCUMENTOS/4.- THESIS/2.- MASTER/1.- DTS/4.- BLUSAS RGB/
3.- TO_MODEL/1.- TRAIN/3.- Y_S/Y_TRAIN.xlsx')
datos_test = pd.read_excel('C:/Users/PAVEL/Documents/
MIS DOCUMENTOS/4.- THESIS/2.- MASTER/1.- DTS/4.- BLUSAS RGB/
3.- TO_MODEL/2.- TEST/3.- Y_S/Y_TEST.xlsx')

##########################################################
##########################################################
##########################################################

num_fotos_train = len(datos_train)
num_fotos_test = len(datos_test)

### Cargamnos las imágenes de TRAIN

B_train=[]
datos_train_1 = datos_train['NOMBRE']

for i in range(num_fotos_train):
#  i=1
ruta = 'C:/Users/PAVEL/Documents/MIS DOCUMENTOS/4.- THESIS/
2.- MASTER/1.- DTS/4.- BLUSAS RGB/3.- TO_MODEL/1.- TRAIN/
2.- GEN/'

ruta_1 = ruta+str(datos_train_1[i])+'.jpg'
fotico = Image.open(ruta_1)
fotico_1 =  np.array(fotico, dtype="float32")
fotico_2 = fotico_1[0:298,0:132,-1]
B_train.append(fotico_2)

### Cargamnos las imágenes de TEST

B_test=[]
datos_test_1 = datos_test['NOMBRE']

for i in range(num_fotos_test):
#  i=1
ruta = 'C:/Users/PAVEL/Documents/MIS DOCUMENTOS/4.- THESIS/
2.- MASTER/1.- DTS/4.- BLUSAS RGB/3.- TO_MODEL/2.- TEST/
```

```
2.- GEN/'

ruta_1 = ruta+str(datos_test_1[i])+'.jpg'
fotico = Image.open(ruta_1)
fotico_1 =  np.array(fotico, dtype="float32")
fotico_2 = fotico_1[0:298,0:132,-1]
B_test.append(fotico_2)

#############################################################
#############################################################

x_train = np.array(B_train, dtype="uint8")
x_test = np.array(B_test, dtype="uint8")

#############################################################
#############################################################

import numpy as np
from tensorflow import keras
from tensorflow.keras import layers

# Model / data parameters
num_classes = 4
input_shape = (28, 28, 1)

input_shape

# the data, split between train and test sets

y_train = datos_train[['T.S','T.M','T.L','T.XL']]

#############################################################
#############################################################

### En esta sección partimos la data en entrenamiento y
 predicción:

y_test = datos_test[['T.S','T.M','T.L','T.XL']]

#############################################################
#############################################################
```

```
#x_all = x_train

#x_train = x_all[0:5000]
#x_train_del = x_all[0:2700]
#x_train_no_del x_all[]
#x_train =
#x_test = x_all[5000:5799]#

#y_all = y_train
#
#y_train = y_all[0:5000]
#y_test = y_all[5000:5799]

############################################################
############################################################

# Scale images to the [0, 1] range

x_train = x_train.astype("float32") / 255
x_test =  x_test.astype("float32") / 255

y_train = np.array(y_train, dtype="float32")
y_test  = np.array(y_test, dtype="float32")

#x_test = x_test_1.astype("float32") / 255
# Make sure images have shape (28, 28, 1)

prueba = np.expand_dims(x_train[1],-1)

x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
#print(x_test.shape[0], "test samples")

############################################################
############################################################

model = keras.Sequential(
[
keras.Input(shape=input_shape),
layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
```

```
layers.MaxPooling2D(pool_size=(2, 2)),
layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
layers.MaxPooling2D(pool_size=(2, 2)),
layers.Flatten(),
layers.Dropout(0.5),
#   layers.Dense(num_classes, activation="softmax"),
layers.Dense(num_classes, activation="relu"),
]
)

model.summary()

batch_size = 128
epochs = 3

##################################################
##################################################

### Escribimos la función de pérdida explicitamente:

import tensorflow as tf
import math

def my_loss(y_true,y_pred):

alfa_1 = tf.slice(y_pred,[0,0],[-1,1])
alfa_2 = tf.slice(y_pred,[0,1],[-1,1])
alfa_3 = tf.slice(y_pred,[0,2],[-1,1])
alfa_4 = tf.slice(y_pred,[0,3],[-1,1])

y_1 = tf.slice(y_true,[0,0],[-1,1])
y_2 = tf.slice(y_true,[0,1],[-1,1])
y_3 = tf.slice(y_true,[0,2],[-1,1])
y_4 = tf.slice(y_true,[0,3],[-1,1])

y_pot_1 = y_1**(alfa_1-1)
y_pot_2 = y_2**(alfa_2-1)
y_pot_3 = y_3**(alfa_3-1)
y_pot_4 = y_4**(alfa_4-1)

epsilon = tf.constant([0.0001])
```

```
gamma_1=tf.math.exp(tf.math.lgamma(tf.math.abs(alfa_1+epsilon)))
gamma_2=tf.math.exp(tf.math.lgamma(tf.math.abs(alfa_2+epsilon)))
gamma_3=tf.math.exp(tf.math.lgamma(tf.math.abs(alfa_3+epsilon)))
gamma_4=tf.math.exp(tf.math.lgamma(tf.math.abs(alfa_4+epsilon)))

#gamma_1 = tf.math.exp(tf.math.lgamma(alfa_1))
#gamma_2 = tf.math.exp(tf.math.lgamma(alfa_2))
#gamma_3 = tf.math.exp(tf.math.lgamma(alfa_3))
#gamma_4 = tf.math.exp(tf.math.lgamma(alfa_4))

gamma_all = tf.math.exp(tf.math.lgamma(alfa_1+alfa_2+alfa_3
+alfa_4+epsilon))

up = y_pot_1*y_pot_2*y_pot_3*y_pot_4

arthur = (gamma_all*up)/(gamma_1*gamma_2*gamma_3*gamma_4)

arthur_1 = -tf.math.log(arthur)

loss = arthur_1
# loss = tf.reduce_sum(-tf.math.log(arthur),axis=0)

return loss

#########################################################
#########################################################

### Compilamos el modelo::

model.compile(loss=my_loss, optimizer="adam", metrics=[my_loss])
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
 validation_split=0.1)

#########################################################
#########################################################

### En esta sección calculamos la función de pérdida de manera
manual ya que se exportan
### libros de excel:

test_loss,test_acc=model.evaluate(x_test,  y_test, verbose=2)
train_loss,train_acc=model.evaluate(x_train,y_train,verbose=2)
```

```
predictions_test = model.predict(x_test)

import pandas as pd

predictions_test_1 = pd.DataFrame(predictions_test)

sumita = predictions_test_1.sum(axis=1)

########################################################
########################################################

### Exportamos los valores de testeo:

import pandas as pd

Datos_testeados_apli = predictions_test_1

Datos_testeados_apli.to_excel(r'C:/Users/PAVEL/Documents/
MIS DOCUMENTOS/4.-THESIS/2.- MASTER/1.- DTS/4.- BLUSAS RGB/
3.- TO_MODEL/2.- TEST/4.-Y_PREDICHO\
Datos_testeados_apli.xlsx', index = False)
```

Bibliografía

Aghdam, H. H. y Heravi, E. J. (2017). *Guide to Convolutional Neural Networks A Practical Application to Traffic-Sign Detection and Classification*, Springer.

Aitchison, J. (1986). *The Statistical Analysis of Compositional Data*, Chapman and Hall.

Bishop, C. M. (1994). Mixture density networks., *Technical report, Aston University* .

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*, Springer.

Bozdogan, H. (1987). Model selection and akaike's information criterion (aic): The general theory and its analytical extensions, *Psychometrika* **52(3)**: 345–370.

Calin, O. (2019). *Deep Learning Architectures. A Mathematical Approach*, Springer.

Casella y Berger (2002). *Statistical Inference*, Duxbury.

Duerr, O., Sick, B. y Murina, E. (2020). *Probabilistic Deep Learning with Python, Keras and TensorFlow Probability*, Manning.

Guigourès, R., Ho, Y. K., Koriagin, E., Bergmann, U. y Shirvany, R. (2018). A hierarchical bayesian model for size recommendation in fashion, *Proceedings of the 12th ACM Conference on Recommender Systems. ACM* pp. 392–396.

Hijazi, R. y Jernigan, R. (2009). Modelling compositional data using dirichlet regression models, *Journal of Applied Probability and Statistics* pp. 77–91.

Karessli, N., Guigoures, R. y Shirvany, R. (2019). Sizenet: Weakly supervised learning of visual size and fit in fashion images, *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* pp. 335–343.

Krizhevsky, Sutskever y Hinton (2017). Imagenet classification with deep convolutional neural networks, *Communications of the ACM* pp. 84–90.

Luce, L. (2019). *Artificial Intelligence for Fashion. How AI is Revolutionizing the Fashion Industry*, Apress, San Francisco.

Maier, M. J. (2014). Dirichletreg: Dirichlet regression for compositional data in r, *Research Report Series/ Department of Statistics and Mathematics 125* .

Pawlowsky-Glahn, V. y Buccianti, A. (2011). *Compositional Data Analysis Theory and Applications*, Wiley.

Song, D., Tong, R., Chang, J., Wang, T., Du, J., Tang, M. y Zhang, J. J. (2017). Clothes size prediction from dressed-human silhouettes, *Next Generation Computer Third International Workshop* pp. 86–98.

Wu, Q., Li, H., Li, L. y Yu, Z. (2019). Quantifying intrinsic uncertainty in classification via deep dirichlet mixture networks.