Clasificación de vocales con redes neuronales

Astrid Giselle Rodríguez Us

Inteligencia Artificial

06 de mayo del 2019

Índice

Introducción

La clasificación es una de las tareas de decisión más comunes en la actividad humana. Un problema de clasificación ocurre cuando un objeto necesita ser asignado a un grupo predeterminado o a una clase por el número de atributos observados relacionados con el objeto. Muchos problemas en negocios, ciencia, industria y medicina pueden ser tratado como problemas de clasificación. Algunos ejemplos específicos son predicción de bancarrotas, diagnóstico médico, control de calidad, reconocimiento de caracteres y reconocimiento de voz.

Las redes neuronales han emergido como una importante herramienta para la clasificación, y poseen diferentes ventajas sobre otros modelos de clasificación. Por ejemplo, las redes neuronales son métodos autoadaptativos impulsados por los datos sin recurrir a ninguna especificación explícita sobre la forma funcional o distributiva de un modelo. Por otra parte,son aproximadores funcionales universales, ya que las redes neuronales pueden aproximar cualquier función con precisión arbitraria. De igual manera, las redes neuronales son modelos no lineales que son flexibles y más adecuados para modelar relaciones complejas del mundo real.

El problema de reconocimiento de caracteres ha sido estudiado y trabajado debido a sus diversas aplicaciones como la digitalización de textos, por mencionar la más común y la cual ha derivado más aplicaciones.

Descripción del problema

Este trabajo tiene como objetivo el reconocimiento de caracteres escritos a mano, en particular de las vocales en minúscula, utilizando algoritmos simples de clasificación no lineal con redes neuronales.

Metodología

Base de datos

La base de datos utilizada para el proyecto fue la *EMNIST: an extension of MNIST to handwritten letters*" (https://arxiv.org/abs/1702.05373v1) .

La famosa base de datos MNIST se derivó de un conjunto de datos más grande conocido como *NIST Special Database 19* que contiene imágenes con dimensiones de 128 x 128 pixeles de dígitos, letras escritas a mano en mayúsculas y minúsculas. Una variante del conjunto de datos NIST completo, denominado MNIST extendido

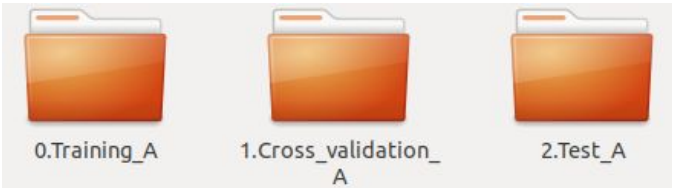
(EMNIST), sigue el mismo paradigma de conversión utilizado para crear el conjunto de datos MNIST. El resultado es un conjunto de conjuntos de datos que constituyen tareas de clasificación más difíciles que involucran letras y dígitos.

Para fines de este trabajo, se procedió a clasificar de manera manual 500 imágenes de cada vocal en minúscula, siendo un total de 2500 imágenes.

Figura 1. Porción de base de datos del EMINST

De manera más específica, se separaron las imágenes por carpetas para cada vocal, y adicionalmente se dividió la cantidad de imágenes para entrenamiento (60%), validación (20%) y pruebas (20%).

a)



b)

Figura 2. a): Clasificación por vocal, b): División de la base de datos.

Por lo consiguiente, se obtuvieron 1500 imágenes para entrenamiento (300 de cada vocal), 500 para validación (100 de cada vocal) y 500 para pruebas (100 de cada vocal). Las imágenes para entrenamiento fueron guardadas en una sola carpeta que incluyera a todas las vocales, sin embargo, se utilizó una nomenclatura de tal manera que las 300 primeras imágenes fueran correspondientes a la letra *a*, las siguientes 300 imágenes fueran correspondientes a la letra *e,* y así sucesivamente hasta llegar a la vocal *u*, esto con el objetivo de poder realizar posteriormente la etiquetación manual para realizar el aprendizaje supervisado. Un procedimiento similar fue realizado para las imágenes de validación y pruebas, con la diferencia de que la cantidad de imágenes por cada vocal fue de 100.

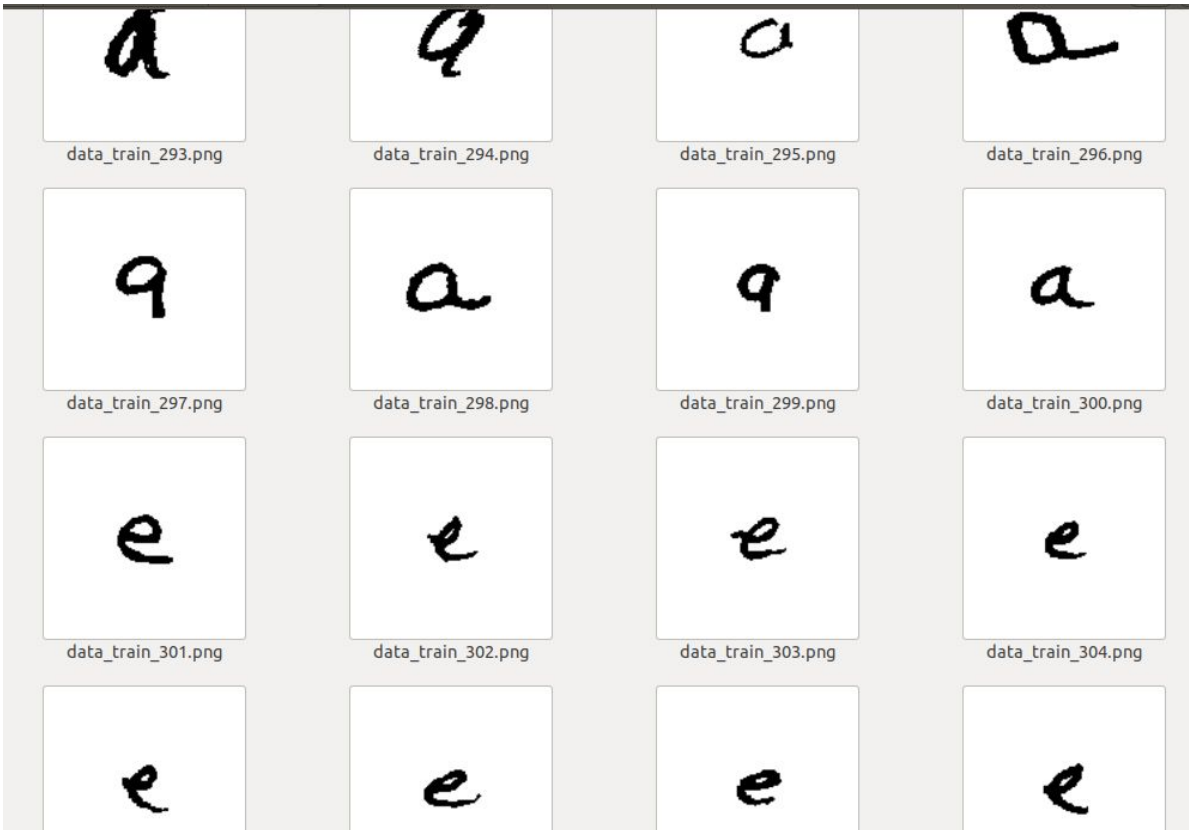


Figura 3. Ejemplo del almacenamiento de las imágenes de entrenamiento. Se puede observar la numeración de las imágenes.

Método

De primera instancia, para el procesamiento de las imágenes, éstas fueron reducidas a un 25% de su tamaño, es decir a 32 x 32 pixeles y convertidas a una escala de grises para así obtener una matriz de sus valores del 0 al 255, siendo 0 el

color negro y el 255 el color blanco. Posteriormente, cada valor fue dividido entre 255.

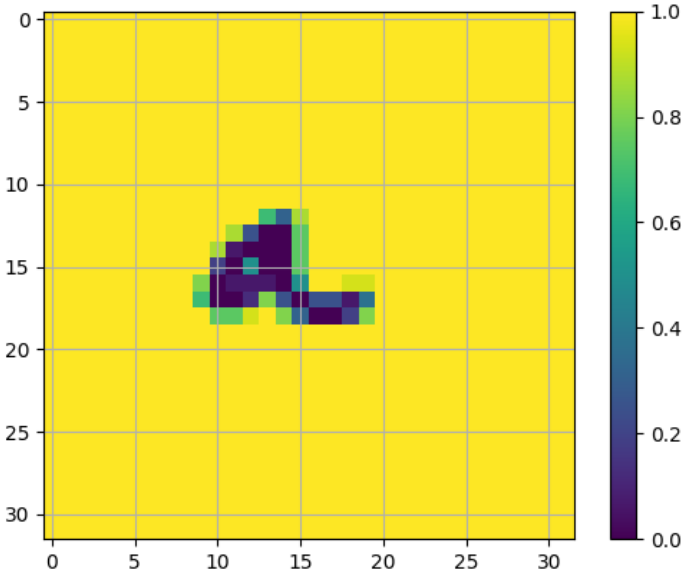


Figura 4. Valores de imagen procesada

Una vez realizada el procesamiento de imágenes, se procedió a realizar el etiquetado con una función en Python. Las etiquetas por cada letra corresponderían a un rango de 0 a 4, donde el número 0 representaría a la vocal a, el número 1 correspondería a la vocal e y la misma analogía para las vocales consecuentes.

Cada matriz de imagen fue guardada como un elemento de una lista, de acuerdo a como habían sido guardadas en su directorio correspondiente. De esta manera, los primeros 300 elementos correspondían a matrices de imágenes de la vocal a, por lo que se creó una lista alterna de etiquetas en la que los primeros 300 elementos fueran el número '0'. Siguiendo la analogía las siguientes etiquetas serían 300 elementos '1' para la letra e, 300 elementos '2' para la letra i, 300 elementos '3 para la letra o y finalmente 300 elementos '4' para la letra u.

```
def getLabelsTRAIN():
    aeiou_label = [0]*300                #a label
    aeiou_label = aeiou_label+[1]*300    #e label
    aeiou_label = aeiou_label+[2]*300    #i label
    aeiou_label = aeiou_label+[3]*300    #o label
    aeiou_label = aeiou_label+[4]*300    #u label
    label_array = np.array(aeiou_label)           #Converts the list into a numpy array
    #print(label_array.shape)
    return label_array
```

Figura 5. Función utilizada para el etiquetado de imágenes

De manera similar se creó una lista cuyo contenido sería la clase de cada imagen, definiendo las clases como ‘a’, ‘e’, ‘i’, ‘o’, y ‘u’.

```
def getClasses_train():
    class_names= ['a']*300                          #a class
    class_names= class_names+ ['e']*300             #e class
    class_names = class_names + ['i'] * 300         #i class
    class_names = class_names + ['o'] * 300         #o class
    class_names = class_names + ['u'] * 300         #u class
```

Figura 6. Función utilizada para la definición de clases de cada imagen.

Finalmente, una vez obtenidas las tres listas con las imágenes, etiquetas y clases, se procedió a unirlas en una sola lista, para de esta manera poder ordenar aleatoriamente los elementos y realizar la fase de entrenamiento.

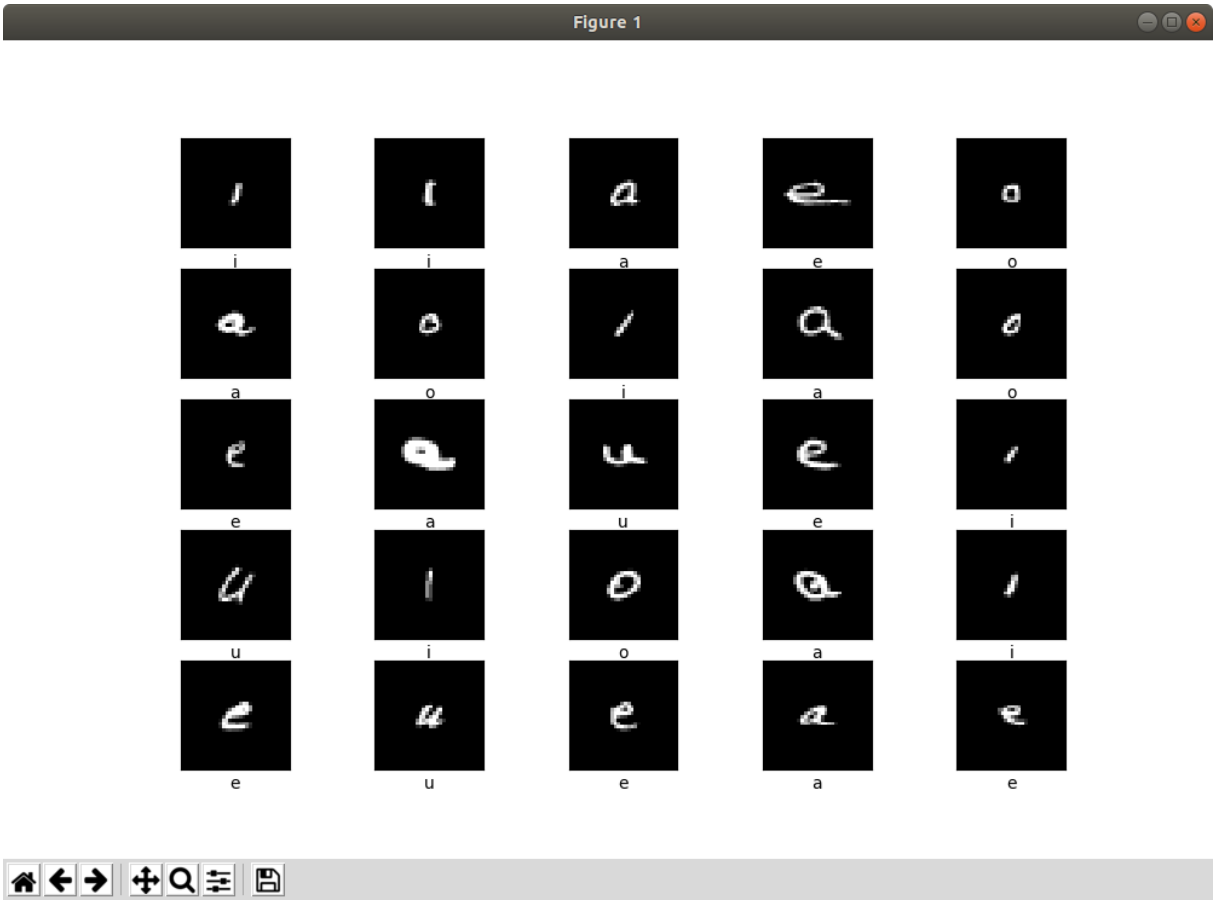


Figura 7. Despliegue de las 25 primeras imágenes con su correspondiente clase obtenidos de lista con los elementos ordenados aleatoriamente

Cabe mencionar que el mismo procedimiento fue realizado para el conjunto de imágenes para validación y para las pruebas, cada una de ellas con su correspondiente etiqueta y clase. Adicionalmente, los conjuntos ordenado aleatoriamente fueron guardados en archivos para posteriormente cargarlos y procesarlos nuevamente.

Arquitectura de la red

Para la construcción del modelo, se utilizó la biblioteca de Tensorflow Keras, una biblioteca de código abierto especializada para desarrollar modelos de aprendizaje automático.

Esta biblioteca contiene un módulo para la implementación de modelos de redes neuronales. En este trabajo se utilizó el modelo secuencial proporcionado por la biblioteca, que es una pila lineal de capas.

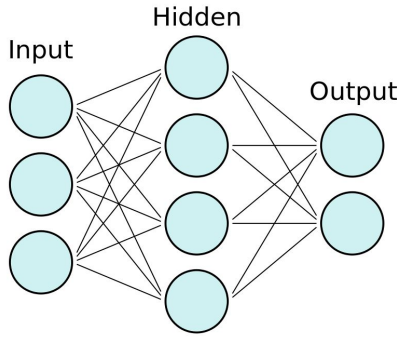


Figura 8. Modelo secuencial de una red neuronal

Para la resolución de este problema, se utilizaron tres capas, una con 1024 neuronas de entrada correspondientes al número de pixeles por imagen, otra capa con 128 neuronas con funciones de activación sigmoidales y por último, una capa de salida con 5 neuronas correspondientes a las 5 clases de vocales y con funciones de activación softmax.

```
def runModel():                                    #Building a simple model

    model = keras.Sequential([
        keras.layers.Flatten(input_shape=(32, 32)),
        keras.layers.Dense(128, activation=tf.nn.sigmoid),
        keras.layers.Dense(5, activation=tf.nn.softmax)
    ])
```

Figura 9. Implementación de la arquitectura de red neuronal con Tensorflow.

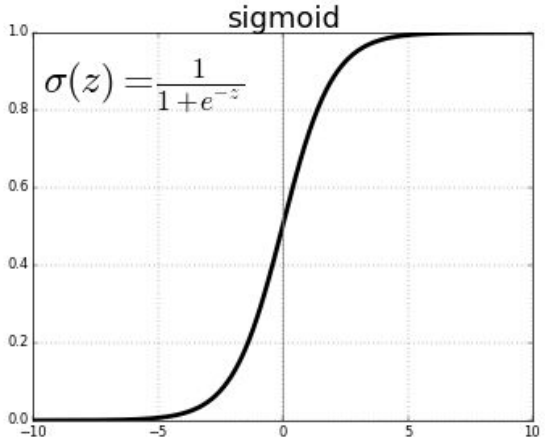


Figura 10. Función sigmoide

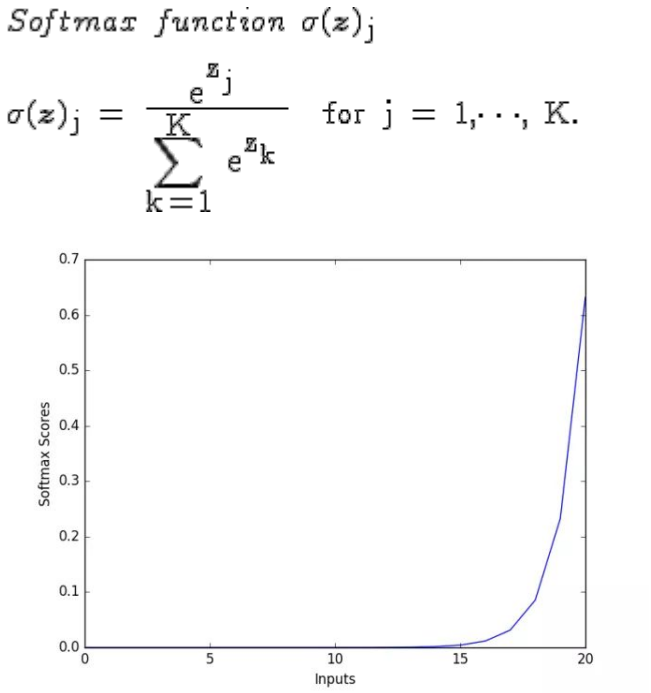


Figura 11. Función softmax

Optimización

Para la funciones de optimización, se eligió la función “Adam”, debido a que fue la que mostró mejores resultados en la minimización del error respecto a otras funciones y la entropía cruzada categórica, la cual mide la media de bits necesarios para identificar un evento de un conjunto de posibilidades.

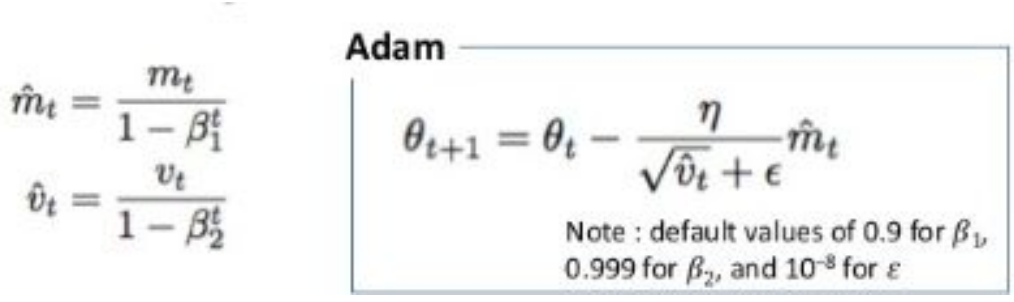


Figura 12. Función de optimización Adam

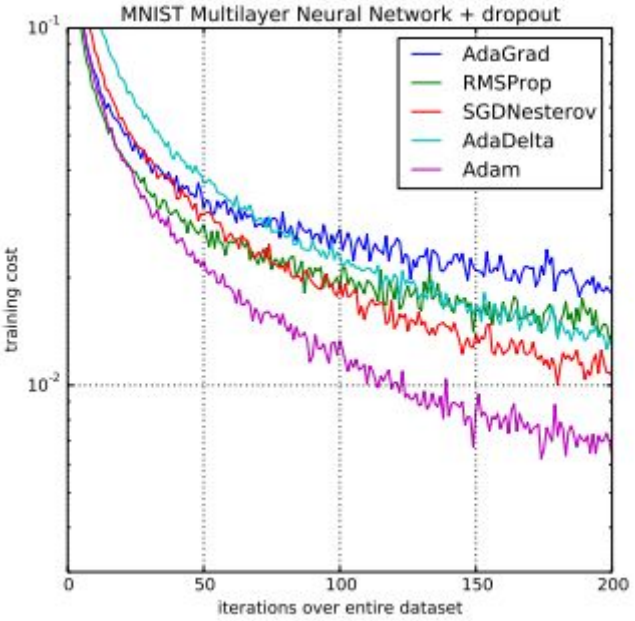


Figura 13. Comparación de funciones de optimización

$$CCE = -\frac{1}{N}\sum_{i=0}^{N}\sum_{j=0}^{J} y_j \cdot log(\hat{y}_j) + (1 - y_j) \cdot log(1 - \hat{y}_j)$$

Figura 14. Entropía cruzada categórica

Pruebas

Se realizaron un total de 500 validaciones (500 imágenes) en las cuales se presentaron resultados de precisión arriba del 90%, en específico con el 93.8% de probabilidad de acierto. Se realizaron en total 14 entrenamientos variando el número de épocas, el tamaño del lote de imágenes así como los tipos de funciones de activación tales como ‘Rectified Linear Unit (ReLU) diferentes tipos de funciones de

optimización, tales como Stochastic Gradient Descent (SGD) y el RMSProp. En todos los casos el grado de precisión fue menor, por ejemplo al utilizar RMSProp se obtuvo un 93.2% de acierto, con ReLU un 90% y SGD con un 88.2 %. De igual manera, al variar el número de lotes para iterar, se notó que al disminuir el número se disminuía la precisión.

Resultados

Gráficas de aprendizaje

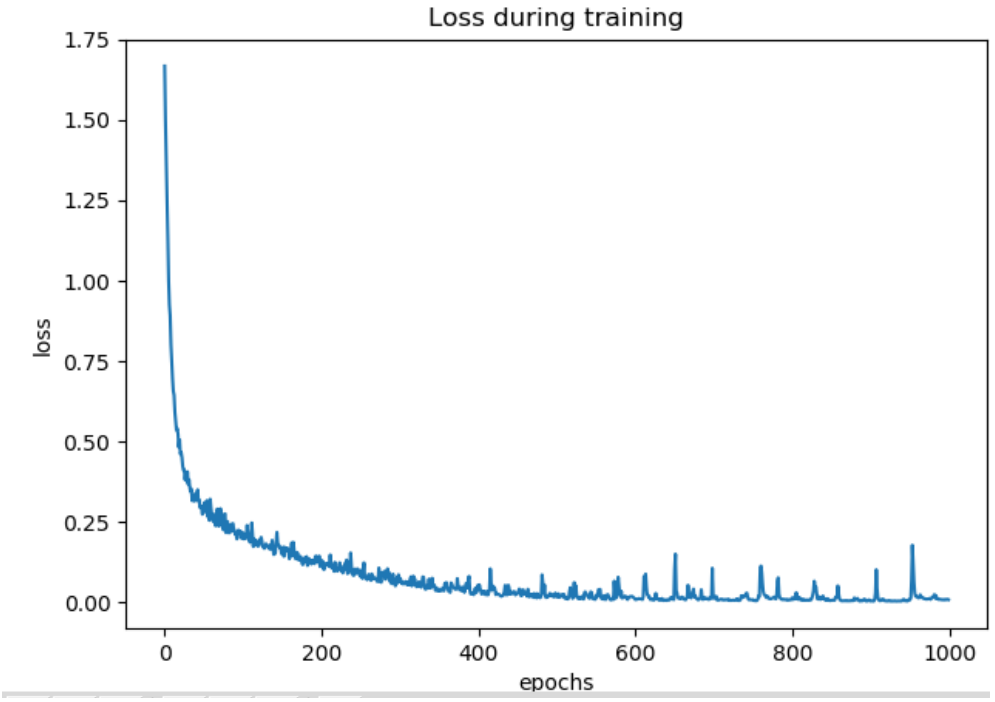


Figura 15. Gŕafica de pérdida durante el entrenamiento.

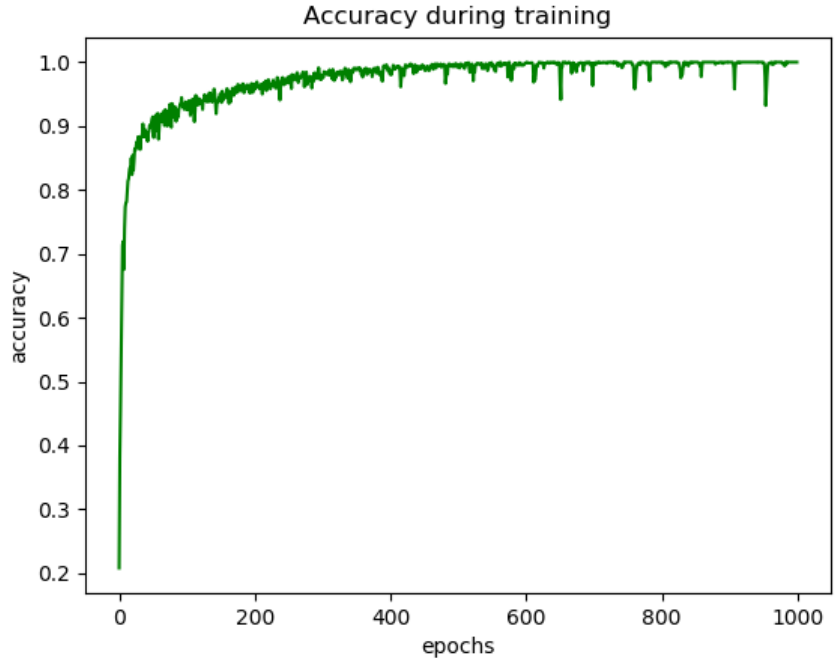


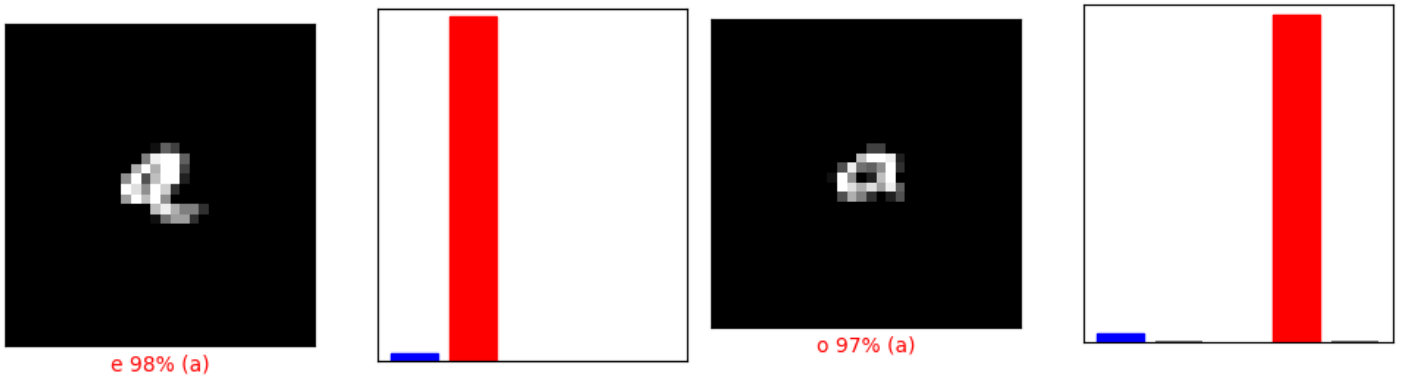
Figura 16. Gŕafica de precisión durante el entrenamiento.

Letra	Número de imágenes	Predicciones correctas	Predicciones incorrectas	Porcentaje de acierto
a	100	84	16	84%
e	100	99	1	99%
i	100	97	3	97%
o	100	99	1	99%
u	100	93	7	93%
Total	500	472	28	
Promedio		94.4	5.6%	94.4%

Tabla 1. Vocal y el resultado de la probabilidad de estimación

Al realizar predicciones con el conjunto destinado a las imágenes de prueba, se obtuvieron los resultados presentados en la Tabla 1, así como también las gráficas de las pérdidas y la precisión durante el entrenamiento.

A continuación se presentan algunos de los verdaderos negativos que se presentaron durante la fase de pruebas.



a) b)

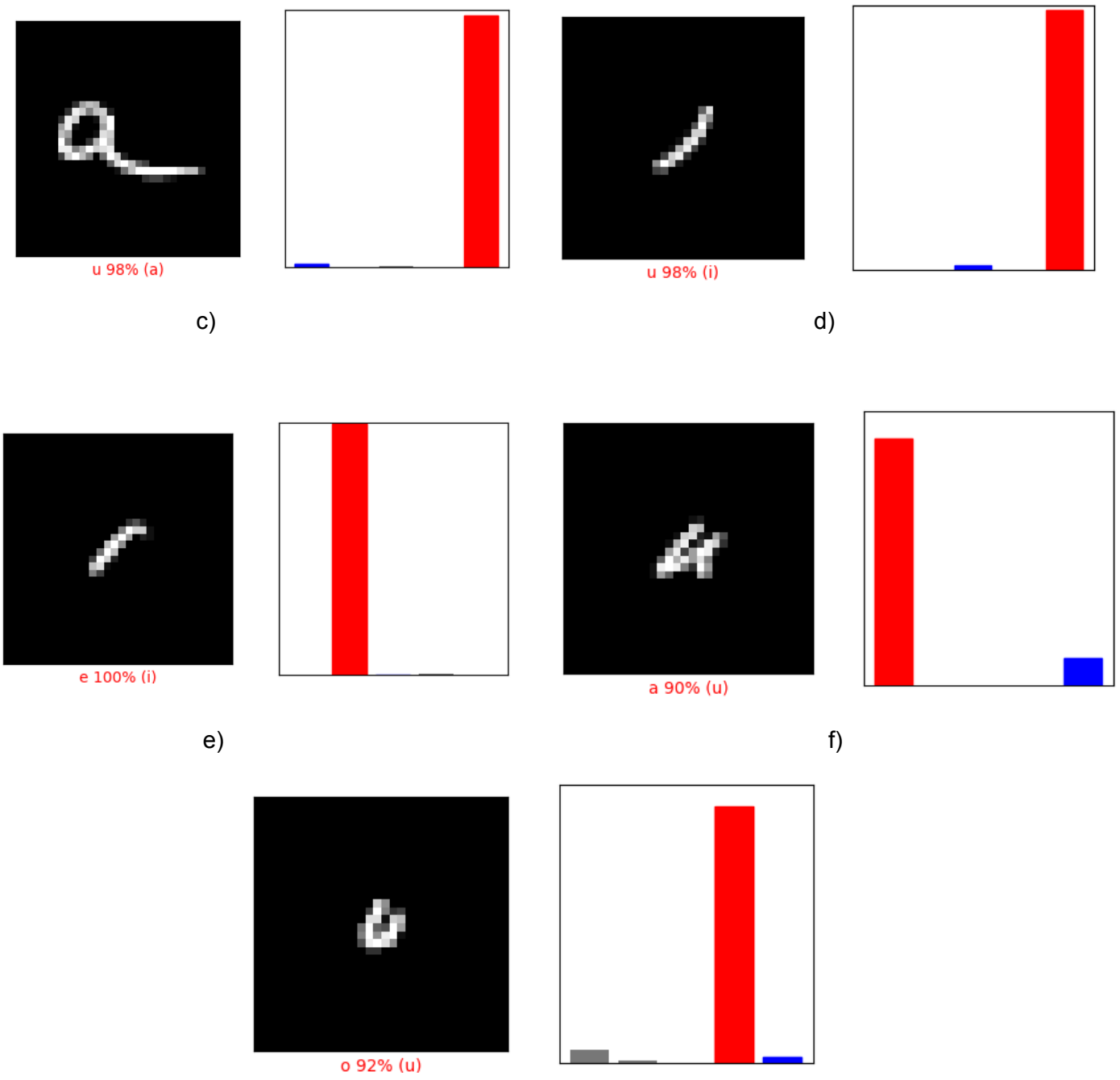


Figura 17. Ejemplos de clasificaciones incorrectas

Falsos Positivos

Durante la realización de pruebas con el conjunto destinado para la tarea, se observó que de todas las clasificaciones realizadas como correctas, no se presentaron en este proyecto falsos negativos, pues al realizarse una verificación de las imágenes y etiquetas, se concluyó que en efecto el algoritmo había acertado el 100% de las veces.

Falsos Negativos

De manera similar a la anterior mencionada, se realizó una verificación de las imágenes arrojadas con una incorrecta clasificación, de las cuales, si bien la gran mayoría fueron correctamente identificadas como mal clasificadas, se presentó el caso en el que se había clasificado correctamente la imagen pero que la etiqueta verdadera había sido incorrectamente colocada, por lo que se descartó como mala clasificación, sin embargo no se consideró como falso negativo ya que no fue error del algoritmo sino del etiquetado.

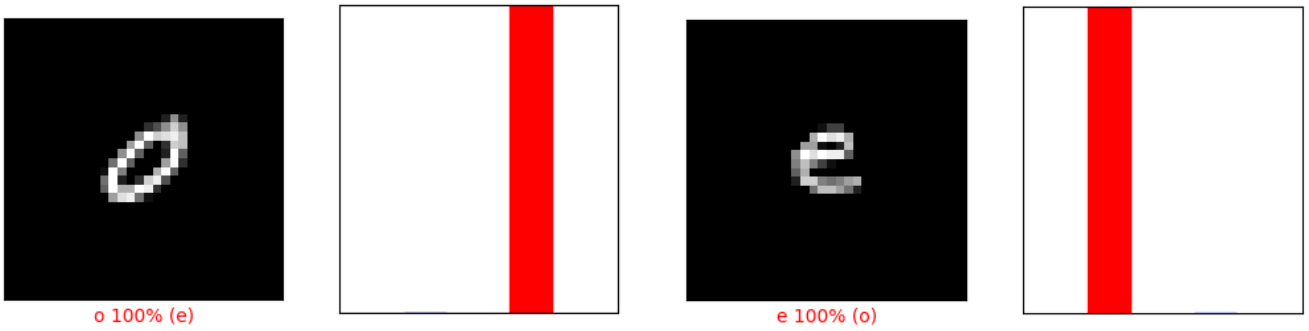


Figura 18. Ejemplos de mala etiquetación

Precisión y exhaustividad

De esta manera, se obtuvieron los siguientes resultados:

Verdaderos Positivos (VP): 472	**Falsos Positivos (FP):** 0
Falsos Negativos (FN): 0	**Verdaderos Negativos (VN)** : 28

Entonces, se obtendría la siguiente precisión:

$$Precisión = \frac{VP}{VP+FP} = \frac{472}{472+0} = \frac{472}{472} = 1.0$$

y de igual manera se obtiene la exhaustividad:

$$Exhaustividad = \frac{VP}{VP+FN} = \frac{472}{472+0} = \frac{472}{472} = 1.0$$

Conclusiones

Las redes neuronales son una importante herramienta para la resolución de problemas del ámbito real, debido en gran medida a su flexibilidad de adaptación. En este trabajo se realizó la clasificación de vocales utilizando como valores de entrada imágenes de dimensiones relativamente pequeñas y monocromáticas las cuales obtuvieron resultados bastante satisfactorios. Es importante recalcar el parecido entre las vocales, por ejemplo la a con la o comparten un círculo, así como de igual manera la a y la u comparten la línea que podría denominarse “cola” . Debido a estas semejanzas y por el tamaño del conjunto de imágenes de entrenamiento, fue difícil incrementar la probabilidad de acierto. Existen otros modelos de redes neuronales que atacan mejor este tipo de problemas, como lo son las redes neuronales convolucionales, así como otros tipos de algoritmos de aprendizaje automático. De esta manera, se pretende continuar con el estudio de algoritmos de aprendizaje.

Referencias

G. P. Zhang, "Neural networks for classification: a survey," in IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 30, no. 4, pp. 451-462, Nov. 2000.doi: 10.1109/5326.897072
URL:
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=897072&isnumber=19421

Train your first neural network: basic classification | TensorFlow Core | TensorFlow
Consultado por última vez 6/05/19:
https://www.tensorflow.org/tutorials/keras/basic_classification

tf.keras.models.Model | TensorFlow Core 1.13 | TensorFlow
Consultado por última vez 6/05/19:
https://www.tensorflow.org/api_docs/python/tf/keras/models/Model

Module: tf.keras.optimizers | TensorFlow
TensorFlow Lite for mobile and embedded devices For Production TensorFlow Extended for end-to-end ML components
Consultado por última vez 6/05/19:
https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

Training and Evaluation with TensorFlow Keras | TensorFlow Core 2.0a | TensorFlow
Consultado por última vez 6/05/19:
https://www.tensorflow.org/alpha/guide/keras/training_and_evaluation

Classification: Precision and Recall | Machine Learning Crash Course | Google Developers
Consultado por última vez 6/05/19:
https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall