UNIVERSIDAD: LA SALLE

LENGUAJE DE SIGNOS A TEXTO

Trabajo Final de Grado

Tutoras:

Núria Valls Canudas y Elisabet Golobardes Ribé

Autora:

Mar Galiana Fernández

Índice

1. Introducción

2. Estado del arte

El análisis del estado del arte, que se realizará en este proyecto, hace referencia a todas las tecnologías y ayudas que disponen las personas sordomudas a la hora de comunicarse con una persona que no sepa el lenguaje de signos.

Esta comunicación se dificulta, no solo por el echo de que la mayoría de la población no tiene los conocimientos necesarios para poder comunicarse en un lenguaje no verbal; sino por el echo de que existen unos 140 lenguajes de signos, con sus respectivos alfabetos y gesto, según la organización Ethnologue [1]. A demás, la mayoría de las personas que nacieron sin la capacidad de oír no saben leer.

2.1. Fundamentos de la enfermedad

2.1.1. Prestación clínica

Los trastornos en el habla afectan a las habilidades comunicativas y pueden ser desarrollarlas por personas de cualquier edad. Hay diferentes tipos, pero los más comunes son: [2]

- Tartamudeo: este trastorno provoca una interrupción en el flujo del habla, que puede ser a partir de repeticiones, bloqueos y/o prolongaciones. [3]
- Apraxia: en la cual la persona no es capaz de realizar acciones cuando se solicitan aunque los músculos requeridos funcionen. Suele ser debido a daños cerebrales, como: un tumor, demencia, una enfermedad neurodegenerativa, lesiones... [4]
- Disartria: en la cual los músculos que permiten el habla se debilitan o se paralizan impidiendo poder pronunciar palabras. La causa suele ser debido a daños cerebrales, tales como: lesiones, tumores, Parkinson, enfermedad degenerativa... [5]

Los síntomas de este trastorno dependen de la gravedad de este, pero en general suelen: repetir o prolongar sonidos, generar sonidos distorsionados, hablar con voz ronca, reorganizar sílabas, agregar sonidos o sílabas a las palabras...[3]

Las posibles causas son [2][3]: daño cerebral debido a un accidente cerebrovascular, cuerdas vocales dañadas, una enfermedad degenerativa como Huntington o Parkinson, cáncer de garganta o de boca, autismo, síndrome de Down, pérdida de la audición...

Hay ciertas características que pueden incrementar las probabilidades de sufrirlo, como pueden ser: [3]

- Ser un hombre.
- Nacer prematuramente.
- Nacer con un peso inferior a lo recomendado.
- Tener antecedentes familiares con este trastorno.
- Tener problemas en partes del cuerpo tales como: la nariz, el oído o la garganta.

TODO: falta la prestación clínica de los sordos

2.1.2. Problemas actuales

- Utilización de teléfonos móviles.
- Las pocas personas que hablan el lenguaje de los signos.
- COVID: las mascarillas impiden leer los labios.

2.2. Aproximaciones actuales

2.2.1. Proyectos realizados

Como se ha comentado, las personas sordo-mudas se enfrentan a diversos problemas, diariamente, ya que muchos de nuestros hábitos no están habilitados para ellos. Es por esto por lo que se han desarrollado diversas tecnologías para poder ayudar a ser una sociedad más inclusiva.

La primera se denomina *Vocalizer To Mute (V2M)*. Es una aplicación que tiene como objetivo facilitar la comunicación entre las personas no hablantes del lenguaje de signo y los sordomudos, especialmente los niños que lo sufren. A partir de la voz de una persona se analiza el mensaje que está transmitiendo y lo representa en el lenguaje de signos con la ayuda de un avatar en 3D. El reconocimiento de voz utiliza la técnica de MFCC (*Mel Frequency Cepstral Coefficients*), la cual permite obtener los componentes más significativos de la señal de audio para poder identificar el contenido relevante, obviando los componentes que aporta información. Se utiliza HTK (*Hidden Markov Model ToolKit*) para poder convertir las señales de los sensores en frases, a partir del módelo HMM la .

The text output obtained from sensor-based system is converted into speech by using the popular speech synthesis technique of hidden Markov model (HMM). The HMM-based-text-to-speech synthesizer (HTS) was attached to the system for converting the text obtained from hand gestures of people into speech

El kit de herramientas HTK se utiliza para convertir estos vectores acústicos en palabras u oraciones reconocibles mediante el uso de un diccionario de pronunciación y un modelo de lenguaje. La aplicación es capaz de reconocer muestras de habla sordomuda de alfabetos en inglés (A - Z), dígitos en inglés (0 a 9) y 15 oraciones comunes que se usan en la vida cotidiana, es decir, buenos días, hola, buena suerte, gracias. , etc. Proporciona servicio de mensajes tanto para sordos como para personas normales. Los sordomudos pueden utilizar un teclado de lenguaje de señas personalizado para redactar el mensaje. La aplicación también puede convertir el mensaje recibido en lenguaje de señas en texto para una persona normal. La aplicación propuesta también se probó en 15 niños de entre 7 y 13 años. La precisión de la aplicación propuesta es del 97,9

2.2.2. Técnicas

2.2.3. Algoritmos de inteligencia artificial

2.2.4. Datos

3. Fundamentos

EXPLICAR:

- Redes neuronales: los diferentes tipos que hay (sobretodo la convolucional), sus diferencias, ventajas y desventajas. ¿Cuáles serian beneficiosas para este problema?

- Árboles de decisión: tipos (sobretodo el Gradient Boosting) y comparativa, ventajas y desventajas. ¿Cuáles serian beneficiosas para este problema?

4. Hardware

A explicar:

- Mac
- Google collab (git)
- Jupyter notebook

Explicar que es cada cosa, ventajas y desventajas

5. Propuesta

Explicación de las diferentes estrategias.

Para la estrategia AccuracyDecisionTree hace falta instalar graphviz para poder mostrar (plot) el modelo de abol de decisión. En el caso de un mac se tiene que hacer con: brew install graphviz

conda install graphviz python-graphviz

https://www.mikulskibartosz.name/how-to-plot-the-decision-trees-from-xgboost-classifier/

https://drive.google.com/file/d/0B0c0MbnP6Nn-eUNRRkVOOGpkbFk/view?resourcekey=0-nVw3WhovKW5FPvPM5GPHfg

6. Experimentación

El objetivo de este proyecto ha sido poder interpretar el lenguaje de signos y mostrar su significado en formato texto, a partir de imágenes. Para poder alcanzar este objetivo se han hecho diversas pruebas y experimentos.

6.1. Experimentación 1: Búsqueda del dataset

En la primera versión se ha utilizado un dataset de la comunidad de datos Kaggle. [6] Consiste en 37 gestos diferentes que representan el abecedario inglés, con las letras de la 'A' a la 'Z', los números del 0 al 9 y el carácter espacio. Cada gesto dispone de 1500 imágenes de 50x50 píxeles. Cada imagen está repetida, ya que hay dos versiones de la misma, una ya preprocesada con la intención de mejorar el entrenamiento y las pruebas de las redes neuronales, y otra versión con las imágenes originales en color. Se han utilizado las dos versiones para poder comparar sus resultados.

A continuación, se muestran diez imágenes que hacen referencia a diez gestos obtenidos del dataset de la la segunda versión mencionada.

Figura 1: Imágenes obtenidas del dataset *Sign Language Gesture Images Dataset* [6] en color.

Como se puede contemplar en la figura anterior, todas las imágenes han sido tomadas con un fondo de un mismo color, centrándose únicamente en la mano que realiza el gesto. La tonalidad de las imágenes cambia dependiendo del gesto, pero todas las imágenes que hacen referencia al mismo, contienen una tonalidad similar.

A continuación, se muestran otras diez imágenes que hacen referencia a los mismos gestos que en la figura anterior, pero estas han sido obtenidas de la primera versión mencionada.

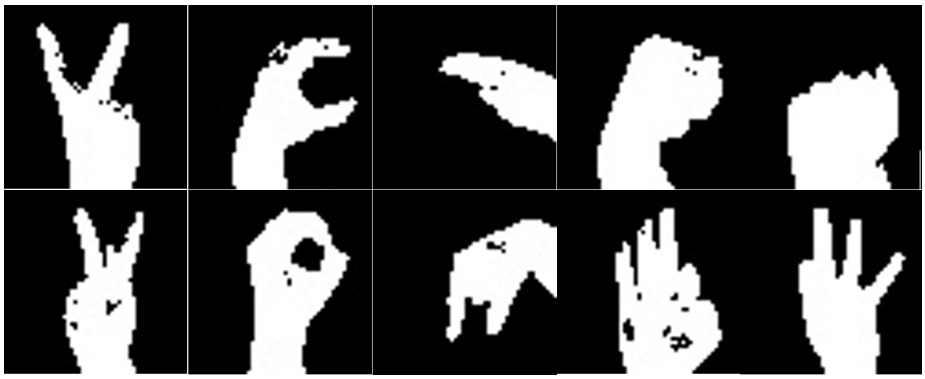


Figura 2: Imágenes obtenidas del dataset *Sign Language Gesture Images Dataset* [6] preprocesadas con la intención de mejorar el entrenamiento de las redes neuronales.

En total, este dataset dispone de unas 55.500 imágenes, englobando las dos versiones.

6.2. Experimentación 2: Uso de redes neuronales

Una vez obtenido el dataset, se decidió empezar utilizando redes neuronales, ya que, como se ha podido apreciar en el estado del arte, es una de las soluciones más recomendadas y utilizadas para el reconocimiento de imágenes.

Se analizaron diferentes tipos de redes neuronales, como se especifica en los fundamentos. Finalmente se decidió empezar por la implementación de una básica y de una convolucional, con el objetivo de poder comparar la precisión de cada una de ellas.

La red neuronal básica se implementó con una estructura de dos capas. Los datos de entrada, que son las imágenes pertenecientes al dataset ya especificado, son previamente procesados, para que sean de una dimensión de 150*150 píxeles. La última capa tienen una dimensión de 38 elementos.

FALTA EXPLICAR BASIC NN

En la figura que hay a continuación, se muestra el resultado del entrenamiento cuando se utiliza el dataset mencionado con las imágenes a color, pero antes de proceder con el entrenamiento son procesadas en blanco y negro, para disminuir el coste de memoria.

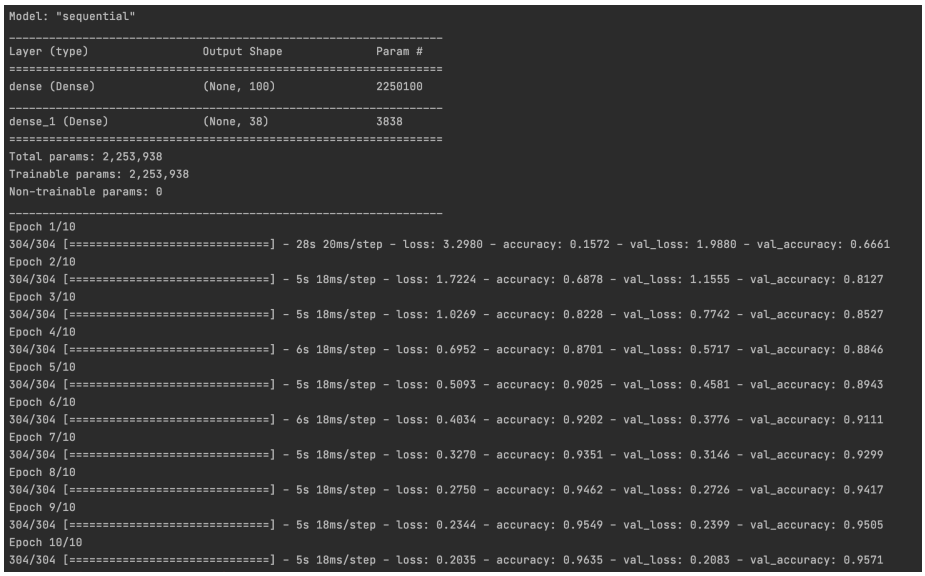
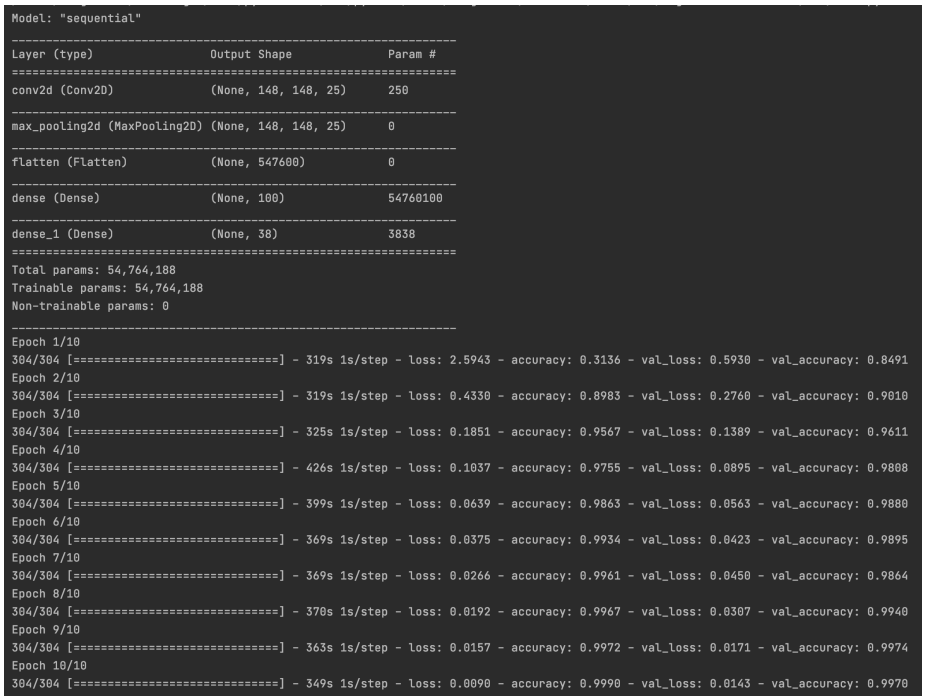


Figura 3: Resultado de la ejecución de la función *summary* con la red neuronal básica.

La red neuronal convolucional fue implementada a partir de cinco capas. La primera tiene dos dimensiones, los inputs son introducidos es una matriz de 150x150 píxeles. La última capa es de una única dimensión de 38 elementos.

FALTA EXPLICAR CNN

La figura que se muestra a continuación contiene el resultado del entrenamiento de la red neuronal convolucional que se acaba de especificar. El dataset utilizado es el que contiene las imágenes a color, pero igual que se ha hecho con la red neuronal básica, las imágenes han sido procesadas en blanco y negro, para disminuir el coste de memoria.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 25)      250
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 148, 148, 25)      0
_________________________________________________________________
flatten (Flatten)            (None, 547600)            0
_________________________________________________________________
dense (Dense)                (None, 100)               54760100
_________________________________________________________________
dense_1 (Dense)              (None, 38)                3838
=================================================================
Total params: 54,764,188
Trainable params: 54,764,188
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
304/304 [==============================] - 319s 1s/step - loss: 2.5943 - accuracy: 0.3136 - val_loss: 0.5930 - val_accuracy: 0.8491
Epoch 2/10
304/304 [==============================] - 319s 1s/step - loss: 0.4330 - accuracy: 0.8983 - val_loss: 0.2760 - val_accuracy: 0.9010
Epoch 3/10
304/304 [==============================] - 325s 1s/step - loss: 0.1851 - accuracy: 0.9567 - val_loss: 0.1389 - val_accuracy: 0.9611
Epoch 4/10
304/304 [==============================] - 426s 1s/step - loss: 0.1037 - accuracy: 0.9755 - val_loss: 0.0895 - val_accuracy: 0.9808
Epoch 5/10
304/304 [==============================] - 399s 1s/step - loss: 0.0639 - accuracy: 0.9863 - val_loss: 0.0563 - val_accuracy: 0.9880
Epoch 6/10
304/304 [==============================] - 369s 1s/step - loss: 0.0375 - accuracy: 0.9934 - val_loss: 0.0423 - val_accuracy: 0.9895
Epoch 7/10
304/304 [==============================] - 369s 1s/step - loss: 0.0266 - accuracy: 0.9961 - val_loss: 0.0450 - val_accuracy: 0.9864
Epoch 8/10
304/304 [==============================] - 370s 1s/step - loss: 0.0192 - accuracy: 0.9967 - val_loss: 0.0307 - val_accuracy: 0.9940
Epoch 9/10
304/304 [==============================] - 363s 1s/step - loss: 0.0157 - accuracy: 0.9972 - val_loss: 0.0171 - val_accuracy: 0.9974
Epoch 10/10
304/304 [==============================] - 349s 1s/step - loss: 0.0090 - accuracy: 0.9990 - val_loss: 0.0143 - val_accuracy: 0.9970
```

Figura 4: Resultado de la ejecución de la función *summary* en la red neuronal convolucional.

Se han implementado tres nuevas estrategias para poder entrenar las redes neuronales especificadas. Estas estrategias permiten guardar los modelos que se utilizan, facilitando la ejecución con diferentes datasets y modelos de redes neuronales, no teniendo que ejecutar de nuevo el procesamiento de las imágenes y el entrenamiento en cada cambio que se realice en el código.

El inconveniente de tener el proyecto separado en estrategias es que se genera un aumento en el coste de memoria, ya que se necesita guardar los cambios realizados en archivos para poder utilizarlos en la siguiente ejecución.

Las estrategias implementadas son:

- **SaveDatabase**: tiene como objetivo procesar el dataset, transformándolo al formato adecuado: cambiando el tamaño y el color de cada imagen. Una vez se han hecho las modificaciones pertinentes, se guarda el modelo en un archivo denominado pickel, con extensión *.pkl*.

  De cada dataset se generarán dos pickels, uno con las imágenes de training y otro con las de testing. Con el dataset con el que estamos trabajando actualmente, en la versión de las imágenes a color, el pickel de testing

tiene un tamaño de 3 GB y el de training de 6,99 GB, 9,99 GB en total.

En el cuadro que se muestra a continuación se puede apreciar la comparación de los tamaños de las dos versiones del dataset cuando se guardan en un directorio, tal y como se descargaron de Kaggle [6], respecto al tamaño del archivo pickel.

	Tamaño directorio	**Tamaño pickel**
Dataset en color	106,7 MB	9,99 GB
Dataset preprocesado	81,6 MB	9.99 GB

Cuadro 1: Comparativa del tamaño del dataset cuando se guardan las imágenes en un directorio, o bien cuando se utiliza un pickel.

Como se puede apreciar en el cuadro anterior, el archivo pickel tiene un tamaño mayor que el propio dataset, pero aún así es beneficioso ya que su lectura es mucho más rápida y nos permite no tener que procesar los datos en cada ejecución. En futuros experimentos se intentará reducir este tamaño con la intención de poder disminuir el coste de memoria.

FALTA COMPARATIVA TIEMPO DE EJECUCIÓN

- **TrainNeuralNetwork**: esta estrategia tiene como objetivo entrenar la red neuronal y guardar el modelo en un archivo con extensión *h5*. Para poder ejecutarla se necesita especificar el tipo de red neuronal que se quiere utilizar. Actualmente hay dos posibilidades: la red neuronal básica y la convolucional. Otro parámetro que también se debe especificar es el pickel con el que se entrenará la red neuronal.

  El modelo se guardará en el archivo *h5* con la intención de poder utilizarlo en otras estrategias, como la que se explicará a continuación, sin tener que volver a entrenarla para poder trabajar con este modelo.

  El tamaño de este archivo depende del tipo de red neuronal utilizada, en este caso el modelo de la básica tiene un tamaño de 27,1 MB, y la convolucional de 657,2 MB, independientemente del pickel utilizado. Esta información se muestra de una forma más gráfica en el siguiente cuadro:

	Tamaño modelo red neuronal básica	**Tamaño modelo red neuronal convolucional**
Dataset en color	27,1 MB	657,2 MB
Dataset preprocesado	27,1 MB	657,2 MB

Cuadro 2: Tamaño del archivo *h5* por los dos tipos de redes neuronales con los dos datasets que se están utilizando actualmente.

El tiempo de ejecución de esta estrategia varia mucho dependiendo del tipo de red neuronal que se esté entrenando y del pickel seleccionado. La red neuronal convolucional puede llegar a tardar alrededor de una hora más que la básica, como se muestra en la tabla que hay a continuación:

	Tiempo entreno red neuronal básica	**Tiempo entreno red neuronal convolucional**
Dataset en color	00:05:54 horas	00:55:45 horas
Dataset preprocesado	00:03:29 horas	01:00:16 hora

Cuadro 3: Tiempo de ejecución de la estrategia de *TrainingNeuralNetwork* dependiendo del dataset y del tipo de red neuronal.

- **AccuracyNeuralNetwork**: Una vez entrenada la red neuronal, se habrá creado un archivo *h5* donde se habrá guardado el modelo entrenado. Esta estrategia procesa el modelo guardado e intenta hacer las predicciones con la parte del dataset definida para el testing, una vez tiene el resultado lo analiza para poder comprobar el número de aciertos. El resultado se mostrará por el terminal.

  En la siguiente tabla se muestra una comparativa del *accuracy* entre las diferentes versiones del dataset: la original y la preprocesada con las dos redes neuronales implementadas.

	***Accuracy* red neuronal básica**	***Accuracy* red neuronal convulocional**
Dataset en color	72.27 %	34.46 %
Dataset preprocesado	100 %	99.96 %

Cuadro 4: Comparativa del *accuracy* de los dos tipos de redes neuronales dado dos modelos con diferente procesado.

Como se aprecia en la tabla 4, la red neuronal básica tiene una *accuracy* mas elevada en los dos Dataset, con lo que se han deducido dos posibilidades:

- No se están introduciendo suficientes datos en la red neuronal convolucional para poder hacer un buen entrenamiento, ya que este tipo de redes necesitan muchos datos para poder aprovechar todo su potencial.

- La configuración con los que se han implementado la red neuronal convolucional no son las más adecuadas para este dataset.

En cuanto al tiempo de ejecución de esta estrategia, si que se encuentra diferencia entre el tipo de red neuronal, pero su duración no llega al minuto, como se muestra la tabla que hay a continuación:

	Tiempo accuracy red neuronal básica	**Tiempo accuracy red neuronal convolucional**
Dataset en color	11 segundos	48 segundos
Dataset preprocesado	15 segundos	43 segundos

Cuadro 5: Tiempo de ejecución de la estrategia de *AccuracyNeuralNetwork* dependiendo del dataset y del tipo de red neuronal.

En futuros experimentos se comprobará si las suposiciones que se han mencionado son acertadas o no, con la intención de mejorar la *accuracy* de las dos redes neuronales.

Una vez analizadas cada una de estas estrategias, podemos comparar el coste global, tanto de memoria como de tiempo. En la siguiente tabla se muestra la suma de los costes de memoria que ha habido en las tres estrategias, dependiendo del dataset y del tipo de red neuronal.

	Coste memoria red neuronal básica	**Coste memoria red neuronal convolucional**
Dataset en color	10,02 GB	10,65 GB
Dataset preprocesado	10,02 GB	10,65 GB

Cuadro 6: Comparativa del coste de memoria en las tres estrategias entre los dos tipos de redes neuronales, dado dos modelos con diferente procesado.

El coste de memoria ha sido calculado a partir de la suma de los archivos que se generan en cada estrategia. En concreto, la suma del tamaño del pickel y del archivo que contiene el modelo entrenado de la red neuronal. Como conclusión, se puede observar como el dataset no supone una diferenciación en el coste de memoria, ya que los pickels tienen el mismo tamaño con los dos procesados.

Lo que provoca la diferencia entre los dos tipos de red neural, es el archivo *h5* que se genera una vez ha sido entrenada y que tiene un tamaño mayor en la convolucional, como queda reflejado en la tabla 6.

En la siguiente tabla se muestra el coste de tiempo de las tres estrategias, sumando el tiempo de ejecución de cada una de ellas, dependiendo del dataset y de la red neuronal utilizada.

	Coste tiempo red neuronal básica	**Coste tiempo red neuronal convolucional**
Dataset en color	00:06:05 horas	00:56:33 horas
Dataset preprocesado	00:03:44 horas	01:00:59 hora

Cuadro 7: Comparativa del coste de tiempo en las tres estrategias entre los dos tipos de redes neuronales, dado dos modelos con diferente procesado.

FALTA SUMAR EL TIEMPO DE LOS PICKELS

En la tabla 7 se puede observar como hay una diferencia abismal entre el coste del tiempo de ejecución de la red neuronal básica a la convolucional, prácticamente la diferencia es de una hora. Con el coste de tiempo sucede lo miso que con el coste de memoria, es indiferente el dataset utilizado. La razón de esto, es que al final hay el mismo número de datos, únicamente cambia el tipo de procesado, el dataset que originalmente es en color, ha sido transformado para poder trabajar en tonalidades grises, demostrando así que no hay un coste adicional de memoria utilizado para guardar la información del color. Se intentó ejecutar las tres estrategias sin transformar la imagen original, para poder comprobar si realmente las imágenes a color tienen una *accuracy* menor que si son en blanco y negro. No se pudo llevar a cabo estas ejecuciones ya que no había memoria suficiente en el dispositivo, pero se probará con otro dataset de dimensiones reducidas.

En cuanto a la conclusión de la implementación de las redes neuronales, se ha observado, en la tabla 4, como la red neuronal básica ha dado mejores resultados que la convolucional, y como el procesado del dataset ha significado un mejora inimaginable en los dos tipos de redes. Esta información se tendrá en cuenta cuando se vaya a utilizar un dataset diferente al actual.

Durante la explicación de cada estrategia se han contemplado diferentes mejoras que se llevarán a cabo en futuras experimentaciones, con la intención de poder obtener el mejor resultado con el mínimo coste posible de las redes neuronales.

6.3. Experimentación 3: Uso de árboles de decisión

Una vez se han obtenido resultados y conclusiones de las redes neuronales, se ha decidido experimentar con otro tipo de estructuras, como son los árboles de decisión.

Como se muestra en los fundamentos, se han analizado diferentes tipos de arboles de decisión. Pero se acabó decidiendo por implementar un Boosted Tree.

EXPLICACIÓN DE LA ELECCIÓN DEL BOOSTED TREE

Se ha utilizado la librería xgboost para poder llevar a cabo la implementación, de donde se ha utilizado la clase XGBClassifier. La imagen que hay a continuación muestra como se ha hecho uso de esta librería parra entrenar el árbol de decisión:

```
xgboost_model = XGBClassifier()
xgboost_model.fit(x_train, y_train, verbose=True, eval_set=[(x_test, y_test)])
```

Figura 5: Uso de la librería xgboost en la implementación del boosted decision tree.

Se he entrenado el árbol de decisión con los dos mismos datasets con los que se han entrenado las redes neuronales, con la intención de poder comparar la *accuracy* final entre las dos estructuras.

Para poder llevar a cabo este entrenamiento se han implementado dos nuevas estrategias, con el mismo objetivo que con las redes neuronales: poder separar la ejecución del entrenamiento de la ejecución de la predicción, con tal de no tener que entrenarlo cada vez que se quiera hacer una predicción. Las estrategias creadas son:

- **TrainDecisionTree**: Esta estrategia tiene el objetivo de entrenar el árbol de decisión dado un pickel que contendrá el dataset con el que se quiere trabajar.

  Una vez entrenado, se guardará el modelo en un archivo con extensión *.pickle.dat*, el cual nos permitirá recuperar dicho modelo para utilizarlo en otras estrategias.

  En el siguiente cuadro, el número 8, se muestra el coste de memoria que genera el archivo donde se guarda el modelo entrenado del árbol de decisión.

Como se puede apreciar, EXPLICAR RESULTADOS, COMPARATIVA ENTRE AMBOS TAMAÑOS.

	Tamaño del modelo del árbol de decisión
Dataset en color	1,1 MB
Dataset preprocesado	FALTA

Cuadro 8: Comparativa del coste de memoria en la estrategia *TrainDecisionTree*, dado dos modelos con diferente procesado.

Para poder evaluar la eficiencia de este modelo predictivo también se debe tener el cuenta el coste de tiempo en su entrenamiento. Como podemos ver en el siguiente cuadro, el tiempo de ejecución mínimo de esta estrategia es de prácticamente cuatro horas, tres horas más que con la red neuronal, como se comparará más adelante. También podemos apreciar que no hay mucha diferencia entre el tiempo de entreno de ambos datasets analizados, pero debemos recordar que ambos pickels, estructura donde se guardan los datasets, tienen el mismo tamaño. ASEGURARSE DE QUE EL TIEMPO ES SIMILAR

	Tiempo ejecución del entrenamiento del árbol de decisión
Dataset en color	03:35:45 horas
Dataset preprocesado	FALTA

Cuadro 9: Comparativa del coste de tiempo en la estrategia *TrainDecisionTree*, dado dos modelos con diferente procesado.

- **AccuracyDecisionTree**: La segunda estrategia que se ha implementado para poder utilizar el aprendizaje basado en árboles de decisión tiene como objetivo mostrar el *accuracy* del modelo entrenado. Se especificara el modelo que se quiere utilizar, el cual será el archivo con extensión *.pickle.dat* que esta estrategia procesará. Una vez se haya leído el modelo se hará una predicción con todos los datos de testig del dataset seleccionado, el cual deberá ser el mismo con el que se hizo el entrenamiento. El tanto por ciento de aciertos se mostrarán por el terminal, informando así del *accuracy* de modelo.

  En el cuadro que hay a continuación se muestran los resultados de esta estrategia, comparando la *accuracy* de los dos modelos entrenados con los dos datasets con los que se han hecho todos los experimentos hasta el momento.

	***Accuracy* del árbol de decisión**
Dataset en color	100 %
Dataset preprocesado	100 %ASEGURARSE

Cuadro 10: Comparativa de la *accuracy* del árbol de decisión, dado dos modelos con diferente procesado.

Como se muestra en el cuadro 10, se obtiene una *accuracy* máxima con ambos datasets, lo que muestra que con este tipo de aprendizaje se llegan a obtener resultados muy óptimos para las condiciones del problema actual que se intenta resolver.

Esta estrategia también muestra, en formato de imagen, la estructura predictiva del árbol de decisión. La estructura definida a partir del dataset a color se muestra en el anexo A.1 y la estructura definida a partir del dataset optimizado, el que ha estado preprocesado con anterioridad, se muestra en el anexo A.2.

EXPLICACIÓN IMÁGENES PLOT

En cuanto a los costes de esta estrategia, nos centraremos en el coste de tiempo ya que no se genera ningún archivo adicional que pudiese incrementar el coste de memoria, el único coste significativo que hay es el propio coste de ejecución.

Se ha ejecutado esta estrategia dos veces, una por cada dataset con el que estamos trabajando, la duración de cada ejecución se muestra en el cuadro que hay a continuación:

	Tiempo ejecución del *accuracy* del árbol de decisión
Dataset en color	00:00:23 horas
Dataset preprocesado	00:00:21 horasASEGURARSE

Cuadro 11: Comparativa del coste de tiempo en la estrategia *AccuracyDecisionTree*, dado dos modelos con diferente procesado.

Se puede apreciar como el tiempo de ejecución de esta estrategia es mínimo con ambos datasets. Se han obtenido muy buenos resultados con este método de aprendizaje: una *accuracy* del 100 % con un tiempo de predicción de unos 20 segundos.

Una vez analizadas las dos estrategias, podemos comparar el coste global de tiempo, ya que el coste de memoria únicamente se ha mencionado el de la primera estrategia, por lo tanto el coste global de memoria será el mismo que el de la estrategia de *TrainDecisionTree*.

En la siguiente tabla se muestra la suma de los costes de tiempo que ha habido en las dos estrategias, dependiendo del dataset utilizado.

	Coste tiempo del árbol de decisión
Dataset en color	1,1 MB
Dataset preprocesado	FALTA

Cuadro 12: Comparativa del coste de tiempo en el uso del árbol de decisión como modelo de predicción dado dos modelos con diferente procesado.

El coste de tiempo ha sido calculado a partir de la suma de los tiempos de ejecución de ambas estrategias. El alto valor de este coste es debido a la primera estrategia, que es la causante de este incremento, la segunda estrategia tenía un coste del orden de segundos. Esto demuestra que el entrenamiento de un árbol de decisión es mucho más costoso que su predicción, deberemos comprobar si, para el problema que queremos resolver en este proyecto, nos puede afectar este coste tan elevado o si es despreciable.

6.4. Comparativa de las experimentaciones 2 y 3

Una vez explicada la experimentación de los árboles de decisión, se puede hacer la comparativa entre los resultados de estos y los de las redes neuronales.

Para poder realizar una comparativa adecuada se deberán utilizar los parámetros que se han ido estudiando durante las experimentaciones: la *accuracy* y los costes, tanto de tiempo como de memoria.

En el cuadro que se muestra a continuación se puede apreciar la comparación de la *accuracy* entre las tres estructuras implementadas: la red neuronal básica, la red neuronal convolucional y el árbol de decisión. Se han utilizado los dos datasets con los que se ha estado trabajando en ambos experimentos.

	***Accuracy* red neuronal básica**	***Accuracy* red neuronal convolucional**	***Accuracy* árbol de decisión**
Dataset en color	72,27 %	34,46 %	100 %
Dataset preprocesado	100 %	99,96 %	100 %

Cuadro 13: Comparativa de la accuracy entre las estructuras implementadas dado dos modelos con diferente procesado.

Podemos apreciar cómo al árbol de decisión es el que tiene una mejor *accuracy*, ya que su valor es del 100 % con ambos datasets. Otra conclusión que podemos obtener es que todas las estructuras tienen un precisión muy elevada cuando se utiliza el dataset específicamente procesado para favoreces el entrenamiento.

En cuanto al coste de tiempo, compararemos la duración de la ejecución del entrenamiento y de la predicción por separado. En el cuadro 14 se pueden ver la duración del entrenamiento en la red neuronal básica, la convolucional y el árbol de decisión, estos valores se han obtenido de las tablas: 3 y 9.

	Duración red neuronal básica	**Duración red neuronal convolucional**	**Duración árbol de decisión**
Dataset en color	00:05:54 horas	00:55:45 horas	03:35:45 horas
Dataset preprocesado	00:03:29 horas	01:00:16 hora	FALTA

Cuadro 14: Comparativa de la duración del entrenamiento entre las estructuras implementadas dado dos modelos con diferente procesado.

En el cuadro 15, se muestra una comparativa de la duración de las predicciones entre la red neuronal básica, la convolucional y el árbol de decisión. Estos valores han sido obtenidos de las tablas: 5 y 11.

	Duración red neuronal básica	**Duración red neuronal convolucional**	**Duración árbol de decisión**
Dataset en color	00:00:11 horas	00:00:48 horas	00:00:23 horas
Dataset preprocesado	00:00:15 horas	00:00:43 horas	00:00:21 horas

Cuadro 15: Comparativa de la duración de la predicción entre las estructuras implementadas dado dos modelos con diferente procesado.

Comparando los valores mostrados en las dos tablas, podemos observar como el árbol de decisión es el que más se deriva, ya que su duración de entrenamiento llega prácticamente a cuadriplica el de las redes neuronales. ASEGURARSE QUE CUADRIPLICA. En cuanto a la duración de las predicciones, todos tienen una duración del orden de segundos, siendo la red neuronal convolucional la que mas se acerca al minuto.

Podemos deducir entonces, que el árbol de decisión es el que nos proporciona una mejor *accuracy*, pero nos genera un mayor coste. En futuras experimentaciones se intentaran mejorar los valores de la red neuronal convolucional, ya que hay diversos parámetros que pueden incrementar su *accuracy* y que podrían obtenerse con un coste menor al árbol de decisión.

6.5. Experimentación 4: Incremento dataset

Se han evaludado varios datasets:
https://www.kaggle.com/kareemalaa74/dataset-asl-test-and-train
https://www.kaggle.com/grassknoted/asl-alphabet
https://www.kaggle.com/lucasvieirademiranda/aslalphabet
https://www.kaggle.com/ammarnassanalhajali/american-sign-language-letters

Finalmente se ha escogido el dataset:
https://www.kaggle.com/kareemalaa74/dataset-asl-test-and-train

motivos: - También incluía números, - Imágenes hechas desde ángulos diferentes a los del dataset actual - Diferentes fondos, en el dataset actual todos los fondos son blancos, en este se han hecho las imágenes en diferentes sitios.

Decision Tree: Accuracy:

```
[INFO]: Strategy selected: --accuracyDecisionTree
[INFO]: Arguments entered:
    * Decision Tree model file: asl_alphabet_gray_150x150px_model.pickle.dat
[INFO]: Accuracy is: 100.00%
[INFO]: Strategy executed successfully
[INFO]: Execution finished
[INFO]: The duration of the execution has been 00:00:12 [hh:mm:ss]
```

Figura 6: Modelo de árbol de decisión al ejecutar la estrategia *AccuracyDecisionTree* utilizando el nuevo dataset en blanco y negro.

Plot decison tree:

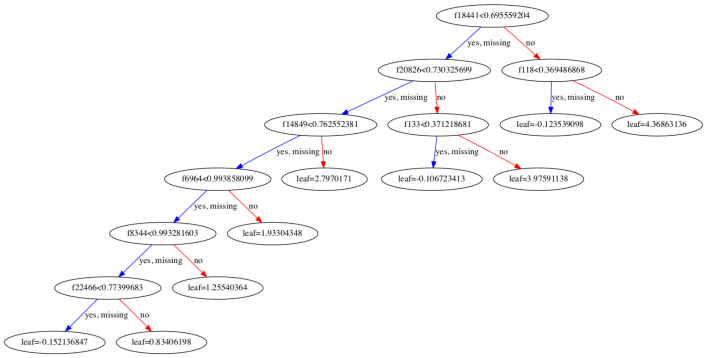


Figura 7: Modelo de árbol de decisión al ejecutar la estrategia *AccuracyDecisionTree* utilizando el nuevo dataset optimizado.

CNN:
TRAIN:

```
In [69]: !python3 Src/main.py --trainNeuralNetwork cnn asl_alphabet_gray_150x150px sign_gesture_gray_150x150px

[INFO]: Strategy selected: --trainNeuralNetwork
[INFO]: Arguments entered:
        * Neural Network type: cnn
        * Pickels selected: asl_alphabet_gray_150x150px, sign_gesture_gray_150x150px
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 25)      250
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 148, 148, 25)      0
_________________________________________________________________
flatten (Flatten)            (None, 547600)            0
_________________________________________________________________
dense (Dense)                (None, 100)               54760100
_________________________________________________________________
dense_1 (Dense)              (None, 39)                3939
=================================================================
Total params: 54,764,289
Trainable params: 54,764,289
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
462/462 [==============================] - 544s 1s/step - loss: 2.0944 - accuracy: 0.5792 - val_loss: 0.7822 - val_ac
curacy: 0.7888
Epoch 2/10
462/462 [==============================] - 510s 1s/step - loss: 0.5132 - accuracy: 0.8591 - val_loss: 0.4022 - val_ac
curacy: 0.8843
Epoch 3/10
462/462 [==============================] - 504s 1s/step - loss: 0.2410 - accuracy: 0.9309 - val_loss: 0.2058 - val_ac
curacy: 0.9421
Epoch 4/10
462/462 [==============================] - 504s 1s/step - loss: 0.1382 - accuracy: 0.9636 - val_loss: 0.1427 - val_ac
curacy: 0.9604
Epoch 5/10
462/462 [==============================] - 501s 1s/step - loss: 0.0799 - accuracy: 0.9809 - val_loss: 0.1115 - val_ac
curacy: 0.9696
Epoch 6/10
462/462 [==============================] - 508s 1s/step - loss: 0.0411 - accuracy: 0.9917 - val_loss: 0.0950 - val_ac
curacy: 0.9754
Epoch 7/10
462/462 [==============================] - 501s 1s/step - loss: 0.0492 - accuracy: 0.9895 - val_loss: 0.0499 - val_ac
curacy: 0.9902
Epoch 8/10
462/462 [==============================] - 509s 1s/step - loss: 0.0162 - accuracy: 0.9979 - val_loss: 0.0430 - val_ac
curacy: 0.9916
Epoch 9/10
462/462 [==============================] - 506s 1s/step - loss: 0.0255 - accuracy: 0.9947 - val_loss: 0.0479 - val_ac
curacy: 0.9895
Epoch 10/10
462/462 [==============================] - 504s 1s/step - loss: 0.0110 - accuracy: 0.9982 - val_loss: 0.0378 - val_ac
curacy: 0.9930
[INFO]: A new decision tree model has been created with the name of: cnn_asl_alphabet_gray_150x150px-sign_gesture_gra
y_150x150px_model.h5
In the path: /Users/margaliana/Documents/Salle/TFG/SignGestureDetection/Assets/ModelStructures/NeuralNetworkModel/
This is the name that will be needed in the other strategies if you want to work with this model.
[INFO]: Strategy executed successfully
[INFO]: Execution finished
[INFO]: The duration of the execution has been 01:35:02 [hh:mm:ss]
```

Figura 8: Modelo de árbol de decisión al ejecutar la estrategia *TrainNeuralNetwork* utilizando el nuevo dataset en blanco y negro.

Accuracy:

```
In [10]: !python3 Src/main.py --accuracyNeuralNetwork cnn cnn_asl_alphabet_gray_150x150px-sign_gesture_gray_150x150px_model.h5

[INFO]: Strategy selected: --accuracyNeuralNetwork
[INFO]: Arguments entered:
        * Neural Network type: cnn
        * Neural Network model file: cnn_asl_alphabet_gray_150x150px-sign_gesture_gray_150x150px_model.h5
[INFO]: Accuracy is: 99.30%
[INFO]: Strategy executed successfully
[INFO]: Execution finished
[INFO]: The duration of the execution has been 00:01:31 [hh:mm:ss]
```

Figura 9: Modelo de árbol de decisión al ejecutar la estrategia *AccuracyNeuralNetwork* utilizando el nuevo dataset en blanco y negro.

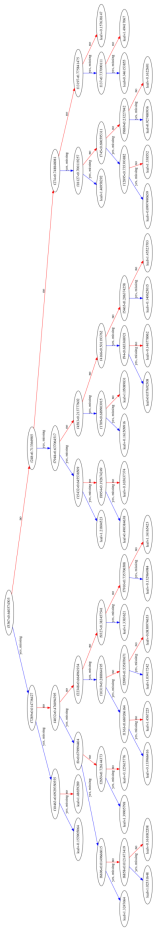
7. Costes del proyecto

8. Conclusiones

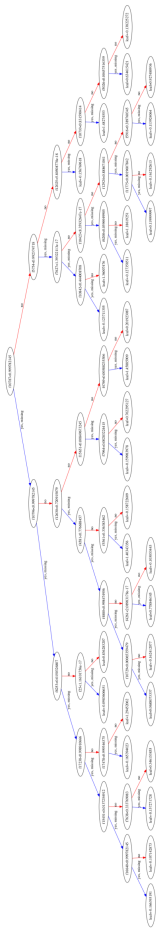
9. Líneas futuras

A. Anexo

A.1. Estructura árbol de decisión con el dataset *Sign Language Gesture Images*



A.2. Estructura árbol de decisión con el dataset *Sign Language Gesture Images* preprocesado



Referencias

[1] Alexey Karpov, Irina Kipyatkovaa, and Milos Zelezny. Automatic technologies for processing spoken sign languages. *Procedia Computer Science*, 81:201–207, 2016.

[2] Sara Minnis. Speech disorders. https://www.healthline.com/health/speech-disorders, September 2019.

[3] Jamie Eske. What are speech disorders. https://www.medicalnewstoday.com/articles/324764, March 2019.

[4] Mazziotta JC Daroff RB, Jankovic J and Pomeroy SL. Apraxia. https://medlineplus.gov/spanish/ency/article/007472.htm, June 2020.

[5] Mazziotta JC Daroff RB, Jankovic J and Pomeroy SL. Disartria. https://medlineplus.gov/spanish/ency/article/007470.htm, August 2020.

[6] Ahmed Khan. Sign language gesture images dataset. https://www.kaggle.com/ahmedkhanak1995/sign-language-gesture-images-dataset, 2019.