# Redes Neuronales Quanvolucionales para Clasificación de Señal Auditiva
# Quanvolutional Neural Networks for Auditory Signal Classification


**Trabajo de Fin de Máster**
**Curso 2023–2024**

**Autor**
**Alejandro Leal Castaño**

**Director**
**Guillermo Botella Juan**

**Colaborador**
**Elías Fernandez-Combarro Álvarez**

**Máster en Ingeniería Informática**
**Facultad de Informática**
**Universidad Complutense de Madrid**

# Redes Neuronales Quanvolucionales para Clasificación de Señal Auditiva
# Quanvolutional Neural Networks for Auditory Signal Classification

**Trabajo de Fin de Máster en Ingeniería Informática**
**Departamento de Arquitectura de Computadores y Automática**


**Autor**
**Alejandro Leal Castaño**

**Director**
**Guillermo Botella Juan**

**Colaborador**
**Elías Fernandez-Combarro Álvarez**


**Convocatoria:** *Septiembre 2024*
**Calificación:** *6.8 (Aprobado)*


**Máster en Ingeniería Informática**
**Facultad de Informática**
**Universidad Complutense de Madrid**

**26 de septiembre de 2024**

# Dedicatoria

A Andújar, Carmen y Chenjie por haber estado siempre ahí desde el principio de la carrera.
A mi abuela por criarme como si fuese otra madre para mí y a mis padres por el apoyo durante todos estos años.

# Resumen


## Redes Neuronales Quanvolucionales para Clasificación de Señal Auditiva

El principal objetivo de este trabajo es el estudio de técnicas de aprendizaje automático cuántico (QML) para tareas de clasificación binaria, dirigidas esencialmente a señal auditiva. En particular nos centramos en las Redes Quanvolucionales o Convolucionales Cuánticas (QCNN), un tipo de modelo de aprendizaje cuántico que toma inspiración en las Redes Convolucionales (CNN) clásicas debido a su bondad para la clasificación y tratamiento de imágenes.

Para ello, se ha definido en primera instancia una serie de QCNNs formadas por distintos circuitos y arquitecturas que serán evaluadas utilizando un dataset de dominio público compuesto por géneros musicales (GTZAN). Posteriormente, se diseña un modelo CNN para comparar la viabilidad de los modelos cuánticos propuestos y señalar las diferencias entre ambos tipos de modelos.

Los resultados presentados aquí muestran unos rendimientos equiparables entre los distintos modelos cuánticos y clásicos, haciendo incapié tanto en la capacidad de aprendizaje como la estabilidad en los entrenamientos. Además, estos resultados también ponen de manifiesto las diferencias en el crecimiento en el número de parámetros cuando se incrementa la complejidad y tamaño de los datos, demostrando las posibles ventajas de estos modelos cuánticos en el fin de la era NISQ.



## Palabras clave

Machine Learning, Quantum Machine Learning, Ansatz, Redes Convolucionales, Redes Quanvolucionales, Computación Cuántica, Circuitos Cuánticos.

# Abstract


## Quanvolutional Neural Networks for Auditory Signal Classification

The main objective of this work is the study of quantum machine learning (QML) techniques for binary classification tasks, primarily focused on audio signals. Specifically, we concentrate on Quantum Convolutional Neural Networks (QCNNs), a type of quantum learning model inspired by classical Convolutional Neural Networks (CNNs) due to their effectiveness in image classification and processing.

To achieve this, we first define a series of QCNNs composed of different circuits and architectures, which will be evaluated using a publicly available dataset of musical genres (GTZAN). Subsequently, a CNN model is designed to compare the viability of the proposed quantum models and highlight the differences between both types of models.

The results presented here show comparable performances between the various quantum and classical models, emphasizing both the learning capacity and stability in training. Moreover, these results highlight the differences in the growth of the number of parameters as the complexity and size of the data increase, demonstrating the potential advantages of these quantum models at the end of the NISQ era.



## Keywords

Machine Learning, Quantum Machine Learning, Ansatz, Convolutional Networks, Quantum Convolutional Networks, Quantum Computing, Quantum Circuits.

# Índice

# Índice de figuras

# Índice de tablas

Capítulo 1

# Introducción

En los últimos años, el desarrollo de la sociedad y la tecnología ha hecho que la inteligencia artificial (IA) adquiera gran importancia, infiltrándose poco a poco en nuestra vida cotidiana. Desde la corrección de textos y la sugerencia de ideas, hasta la identificación de defectos en piezas móviles, los modelos de aprendizaje automático están cada vez más presentes en diversos aspectos de nuestra vida, tanto en el trabajo como en el ocio y las redes sociales.

La velocidad del avance tecnológico y la cantidad de información generada por la humanidad en su evolución como especie está provocando que cada vez sea más difícil tratar con todos estos datos. Junto a este crecimiento, los modelos de aprendizaje automático necesitan seguir mejorando y aumentando su complejidad, utilizando distintos métodos para procesar los datos, con el objetivo de extraer relaciones entre ellos y generalizar patrones a partir de toda esa información. A su vez, esto necesita de potentes ordenadores con gran capacidad de cómputo para el entrenamiento de dichos modelos de IA, suponiendo un reto para el continuo desarrollo del Machine Learning (ML) tradicional. Por tanto, es necesario ampliar nuestras posibilidades y estudiar otros tipos de modelos más eficientes u otros paradigmas de computación. Algunos de estos paradigmas pueden ser la computación analógica, la computación óptica o fotónica o la computación cuántica (QC).

Esta última se trata de una pieza central de este trabajo, aportando valiosas ventajas que nacen de la propia naturaleza de la física cuántica. El entrelazamiento, la interferencia y la superposición de estados permiten el tratamiento de la información de una manera más rápida y escalable que la computación tradicional, en la que nos estamos empezando a encontrar limitaciones físicas en el hardware, con el fin de la ley de Moore y el límite físico en el tamaño de los transistores.

En consecuencia, la principal motivación para investigar y tener en cuenta la QC, es el de dar alternativas a los modelos clásicos actuales y estudiar nuevas ideas de ML inspirados en este nuevo paradigma, ya que promete mejoras teóricas significativas con relación a tiempos de ejecución y complejidad algorítmica. Esto nos lleva a intentar aprovechar este potencial en otro tipo de tareas, ya que la cantidad de información que puede codificarse en un conjunto de qubits[1] es exponencialmente mayor que la que se puede encontrar en un conjunto de bits.

Además, también nos podemos beneficiar de las propiedades de paralelización que presentan los circuitos cuánticos y la capacidad para poder trabajar con mayor cantidad de datos al mismo tiempo gracias al entrelazamiento, pudiendo trabajar con gran cantidad

[1] Unidad de información cuántica. Acrónimo de quantum bit.

de características en los datos en proporción a los recursos disponibles. Por otro lado, la expresividad de los estados cuánticos puede aportar un mejor aprendizaje con una reducida cantidad de información.

El desarrollo de la computación cuántica está liderado por dos frentes: hardware cuántico y diseño de algoritmos o circuitos cuánticos. A pesar de los rápidos avances, el componente hardware se encuentra con algunos inconvenientes como, por ejemplo, el impacto del ruido, la decoherencia cuántica y la dificultad para mantener estados cuánticos durante periodos prolongados de tiempo. Por tanto, se mueve con un ritmo más lento que el desarrollo de algoritmos cuánticos.

Continuando con la exploración de alternativas a los modelos tradicionales, surge un posible competidor para la ejecución de estas tareas: el aprendizaje automático cuántico (QML, por sus siglas en inglés). Así, y aunque todavía los ordenadores y circuitos cuánticos actuales presenten problemas importantes de implementación, ya se han desarrollado tanto algoritmos cuánticos como arquitecturas cuánticas inspiradas en modelos de Machine Learning clásico. Una de estas arquitecturas y el núcleo del estudio de este trabajo son las Redes Convolucionales Cuánticas o Redes Cuanvolucionales (Quantum Convolutional Neural Network o QCNN). Es por ello por lo que este estudio se basa parcialmente en los modelos presentes en [8]. Sin embargo, nuestro trabajo explora nuevas arquitecturas, dirigiendo nuestra atención en su evaluación. También se hace un análisis comparativo exhaustivo respecto a sus contrapartes clásicas, las redes convolucionales (CNNs). Este tipo de modelos ataca uno de los principales problemas del entrenamiento de circuitos presente únicamente en QML y que comúnmente se le conoce por el nombre de *barren plateaus* [4, 5, 9]). Este problema puede dificultar enormemente la actualización eficiente de los parámetros, debido al "paisaje plano" que pueden presentar las funciones de coste en estos sistemas. Además, la codificación de las imágenes en estados cuánticos permite el entrelazado masivo de los datos, haciendo que el modelo pueda extraer relaciones y características que de manera tradicional no sería posible.

Con estas ideas presentes, los principales objetivos de este trabajo son los siguientes:

- Aprender con mayor profundidad el funcionamiento del campo del Quantum Machine Learning, interpretar resultados de los experimentos propuestos y extraer observaciones y conclusiones sobre estos resultados.

- Explorar las posibilidades que ofrece la biblioteca de construcción de circuitos `Hierarqcal`, reproducir los resultados del artículo que lo presenta y aportar ideas de mejora en los modelos que se proponen.

- Comparar los resultados entre las distintas arquitecturas y modelos. De esta manera se puede observar el impacto del circuito y el número de parámetros en el entrenamiento y aprendizaje de los modelos cuánticos.

- Comparar la efectividad de modelos de Machine Learning clásicos y cuánticos, con la intención de mostrar la potencia de los modelos cuánticos frente a los clásicos, fijándonos en la complejidad y los parámetros entrenables de cada uno de ellos.

- Comprobar la importancia e influencia de los datos y sus características en el entrenamiento de los modelos y en sus resultados.

Para terminar con esta introducción, este trabajo se organiza de la siguiente manera en los próximos capítulos.

En el capítulo 2 se da una visión general del estado del arte, incluyendo la era NISQ, las QCNNs, la eficacia de los sistemas y modelos en cuanto a tareas de clasificación de imagen y audio.

En el capítulo 3, se presenta gran parte de la extensión del trabajo realizado, explicando con detalle la selección de las arquitecturas de los modelos cuánticos y clásicos, así como el código utilizado para su definición y construcción. También se incluye en este capítulo la metodología general y de entrenamiento de dichos modelos.

El capítulo 4 muestra toda la variedad de experimentos realizados, junto los resultados obtenidos y algunas observaciones que se pueden extraer de estos resultados. En el capítulo 5 se presentan las conclusiones que se pueden deducir de los resultados de nuestro trabajo, junto con posibles líneas de trabajo futuro.

Por último, los capítulos 6 y 7 son las versiones en lengua inglesa de la introducción (capítulo 1) y de las conclusiones y trabajo futuro (capítulo 5), respectivamente.

Capítulo 2

# Estado del arte

En este capítulo se abordará el contexto de la era NISQ (Noisy Intermediate-Scale Quantum) y las limitaciones actuales del hardware cuántico, tales como el ruido y la falta de corrección de errores. Además, se presentarán los principales avances en el desarrollo de algoritmos cuánticos, con especial atención a la aparición del Quantum Machine Learning (QML). Finalmente, se explorarán los frameworks y simuladores disponibles para la implementación y ejecución de circuitos cuánticos, tanto en simulación local como en hardware real.

## 2.1. Era NISQ

Desde los años 80, con la primera introducción de la computación cuántica por Richard Feynmann y el diseño de los primeros algoritmos cuánticos, este campo se ha visto en constante cambio e innovación, mostrando grandes avances y prometiendo mejoras significativas en tiempo de ejecución y capacidad de cálculo. Todo ello gracias al potencial de las propiedades que sólo aparecen en este escenario cuántico.

Pero a pesar de esta rápida evolución, todavía nos encontramos con numerosos retos y obstáculos a la hora de construir un sistema que presente estas propiedades físicas que las hacen tan deseables, con lo que la comunidad científica ha establecido términos que capturen la realidad actual en lo que se refiere a estos avances y dejar definidos objetivos a futuro.

La era NISQ (por sus siglas en inglés, Noisy Intermediate-Scale Quantum) es un término que hace referencia a la etapa del desarrollo de los circuitos y hardware cuántico del que se dispone en la actualidad. Estos sistemas presentan 2 tipos de limitaciones:

- "Noisy" hace referencia a la influencia del "ruido" que provoca que el estado cuántico este sujeto a interferencias que lo alteran, provocando errores en las operaciones que este circuito realice. Aquí también se incluye el problema de la decoherencia, en el que 2 (o más) qubits entrelazados pueden "desentrelazarse", perdiendo información del estado cuántico inicial y la aparición de más errores de cálculo.

- "Intermediate-Scale" indica el tamaño o complejidad que estos ordenadores pueden presentar. Aunque el término no expresa explícitamente este tamaño, la comunidad científica suele referirse a circuitos entre 50 y 100 qubits. Actualmente, la tecnología ha seguido avanzando de tal manera que algunos dispositivos NISQ alcanzan los 400 qubits. Además, si esto se relaciona con el punto anterior y los ordenadores cuánticos

afectados por el ruido, esto también pone en contexto que estos circuitos no tienen implementado una corrección de errores cuánticos (usualmente denominado en la literatura como Quantum Error Correction o QEC), ya que es necesario tanto de disponer de más qubits como de tener suficiente profundidad en los circuitos, puertas cuánticas y recursos de hardware (lo que provoca a su vez mayor impacto del ruido en el sistema).

A pesar de todos los avances actuales, todavía queda un largo recorrido para disponer de implementaciones de qubits robustas y fiables, que nos permitan verificar en la práctica las ventajas que la computación cuántica ofrece frente a la computación clásica binaria.

## 2.2. Algoritmos cuánticos y aparición del QML

Un primer uso que se le ha dado a los circuitos cuánticos es el de intentar aprovechar las ventajas de este nuevo paradigma con el objetivo de obtener mejores soluciones a problemas clásicos, notablemente conocidos por ser intratables por los algoritmos actuales cuando el tamaño del problema crece. Algunos de estos problemas pueden ser el problema del viajante o de la mochila, cuyas soluciones óptimas no pueden ser ejecutadas en tiempo polinomial en un ordenador clásico y, por tanto, no pueden ser solucionados en un tiempo que se pueda considerar razonable. Estas estrategias están orientadas no tanto al diseño de algoritmos o implementación en pseudocódigo, sino al diseño de circuitos que permitan solucionar el problema, interpretando los estados cuánticos de una manera adecuada. Todas estas ideas se implementan con el uso de puertas lógicas cuánticas como si se tratasen de puertas lógicas clásicas que modifican bits en un ordenador tradicional.

De esta manera, surgen dos de los principales algoritmos cuánticos ampliamente conocidos en el mundo de la algoritmia cuántica: el algoritmo de Grover y el algoritmo de Shor, tal y como se ilustra en la figura 2.1.

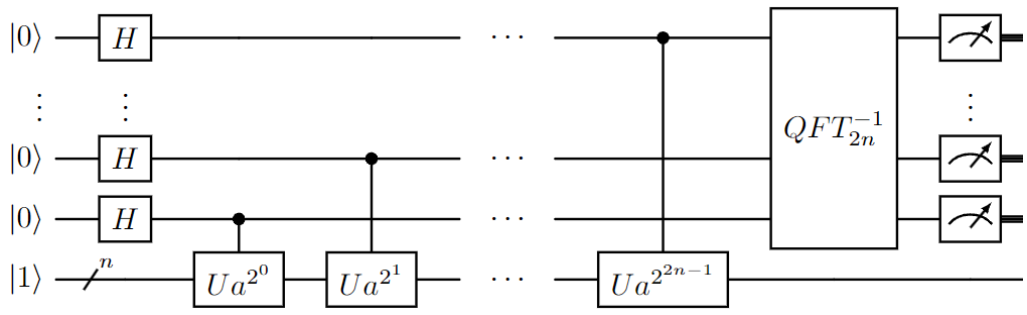


Figura 2.1: Algoritmo de Shor implementado en un circuito cuántico.

Si nos detenemos un momento aquí, los algoritmos anteriormente descritos solo están concebidos para realizar únicamente esta tarea, con lo que el circuito que resuelve estos problemas es siempre el mismo. La existencia de puertas lógicas cuánticas parametrizadas por valores reales (como pueden ser las puertas de rotación) nos permiten ir más allá y construir circuitos con mayor versatilidad y expresividad. Este tipo de circuitos también se denominan como Parametrized Quantum Circuits (PQC), debido a la capacidad de cambiar su comportamiento con la variación de sus parámetros y permitiéndonos aprender de los datos de entrenamiento.

Con estas ideas presentes, se pueden definir distintos tipos de QML en función del tipo de datos que se utilice (datos clásicos o cuánticos) o el tipo de modelo o entrenamiento

que se desee entrenar (modelo clásico o circuitos cuánticos). La combinación en la que se centra este trabajo se trata del uso de datos clásicos junto con circuitos cuánticos. Este tipo de procedimiento también se denomina comúnmente como entrenamiento híbrido (o hibridación) debido a que tiene 2 etapas bien diferenciadas en la que forman parte tanto componentes cuánticos como clásicos. En cada iteración de entrenamiento, la primera etapa consiste en la ejecución del circuito o modelo cuántico (que puede llevarse a cabo mediante simulación o por hardware real). La segunda etapa se realiza en un ordenador clásico o clásicamente: Se recopilan los outputs que se obtienen del circuito anterior que servirán como entrada para un optimizador (por ej. se calculan los gradientes), se evalúa la función de pérdida y se actualizan los parámetros del modelo para volver a ser ejecutados en la siguiente iteración. La figura 2.2 muestra un esquema entre la relación del circuito con el ordenador clásico que optimiza sus parámetros.

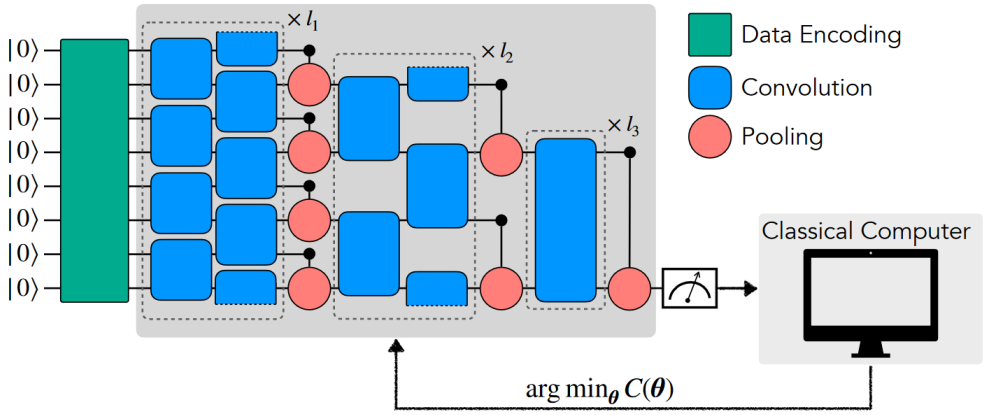


Figura 2.2: Entrenamiento híbrido dividido en dos partes: una cuántica con la ejecución del modelo y una clásica con entrenamiento en un ordenador clásico que se actualiza los parámetros. Fuente: [5].

Por otro lado, mientras que se han desarrollado maneras de entrenar PQCs sin la necesidad de un ordenador clásico que actualice los parámetros [7], estas ideas requieren de una gran cantidad de qubits, por lo que actualmente no resulta ser factible ni por simulación ni por hardware, debido a la era NISQ.

Al tratarse de un campo en desarrollo, los principales circuitos y modelos de QML no están todavía asentados y se encuentran todavía en etapas de investigación y evolución, donde no se tiene claro que estrategias son las mejores para cada tipo de problema. Sin embargo, se han estado proponiendo numerosas metodologías y modelos cuánticos inspirados en ideas tradicionalmente clásicas como, por ejemplo, el dropout cuántico [10], el *ensemble learning* [6] o las redes neuronales convolucionales cuánticas [3, 4, 5, 8]:

**Dropout cuántico:** Análogo directo al dropout tradicional. En este caso, el dropout cuántico consiste en suprimir algunas puertas del circuito en cada iteración del entrenamiento, pudiendo definir así distintas maneras de realizar el dropout en función del tipo de puerta lógica cuántica que se omita. El principal objetivo consiste en minimizar el overfitting y entrenar neuronas más robustas e independientes entre sí. La figura 2.3 muestra un esquema del funcionamiento de esta estrategia.

**Ensemble learning:** Este tipo de estrategia se basa en el concepto de inteligencia colectiva. Consiste en el uso de multitud de modelos simples y poco potentes, pero que

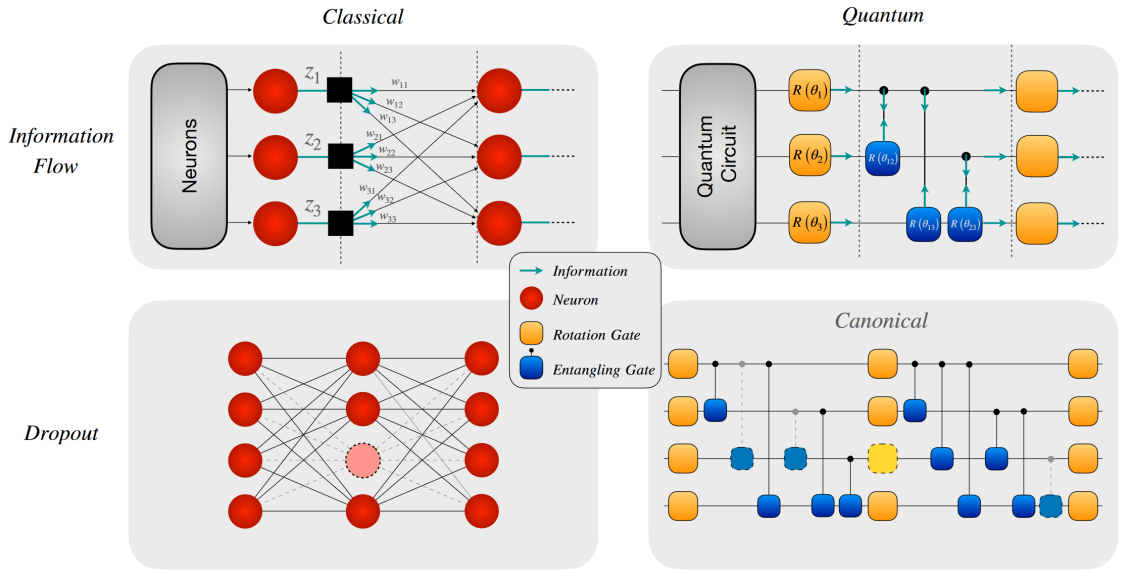


Figura 2.3: Comparación entre el flujo de información clásico y cuántico (arriba) y comparación entre el dropout clásico y dropout cuántico (abajo). Fuente: [10].

trabajan en conjunto, de tal manera que, como grupo, pueden superar en algunos casos la capacidad de clasificación de un único modelo más complejo y expresivo. La manera de entrenar cada modelo depende del método que se determine para combinar cada una de las distintas predicciones. Este tipo de modelos se han estudiado principalmente para el ahorro de recursos, robustez en el entrenamiento y la efectividad frente a las *barren plateaus*, debido al reducido número de qubits.

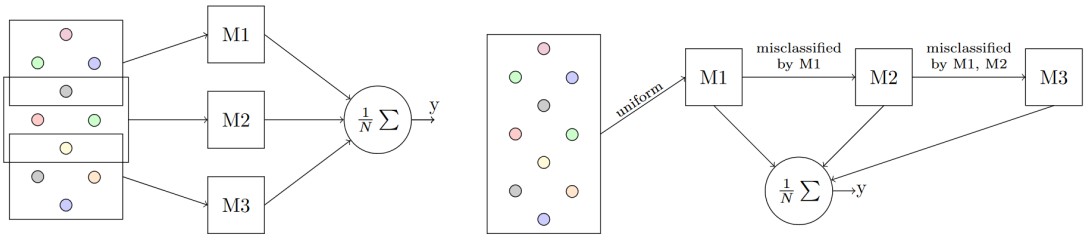


Figura 2.4: Esquema de funcionamiento con ensemble learning para técnicas de *bagging* (izquierda) y *stacking* (derecha). Fuente: [6].

**QCNN:** Las QCNNs son la variante cuántica de las CNNs. A pesar de su nombre, no existe un circuito cuántico que realice una operación totalmente análoga a las convoluciones. Sin embargo, se han propuesto múltiples alternativas que puedan ocupar este papel, construyendo arquitecturas que presenten invarianza translacional. Esto se consigue aplicando un ansatz de manera uniforme en todos los qubits. Se puede ver este ansatz como el equivalente a un kernel en una convolución de una red convolucional tradicional.

Además, el desarrollo del Machine Learning clásico ha propiciado el paso de la selección manual de características a uno totalmente automático con la aparición del Deep Learning (DL) y las Deep Neural Networks (DNN). Siguiendo con esta evolución, el siguiente paso

ha consistido en la creación automática de modelos, que en algunos casos ha podido generar redes con una efectividad cercana al estado del arte [1, 15]. Sin embargo, a pesar de la potencia que estos métodos pueden ofrecer, estas técnicas han sido poco exploradas debido a la inmensa intensidad de cálculo que requiere. Esta ha sido una de las principales motivaciones de la creación de la librería `Hierarqcal`, que permite la construcción de circuitos y arquitecturas cuánticas de manera jerárquica y versátil, orientado a facilitar las tareas de los que se conoce como Neural Arquitectural Search (NAS)[1].

## 2.3. Frameworks, simuladores y ejecución en hardware

Una vez comentado el contexto en el que nos encontramos respecto al hardware y algunas de las técnicas y modelos que se están desarrollando en el Machine Learning cuántico, es necesario señalar algunas de las alternativas que tenemos disponibles a la hora de construir y ejecutar circuitos y algoritmos cuánticos.

En primer lugar, en cuanto a la emulación de circuitos cuánticos, existen alternativas tanto para simulación local como la utilización de herramientas y simuladores en la nube. Los backends más populares para simulación se encuentran disponibles como librerías de Python, `pennylane` y `qiskit`, ofreciendo múltiples dispositivos de emulación basados principalmente en vectores de estado. La simplicidad y su fácil uso para construir estos circuitos es lo que los convierte en los principales frameworks para la creación y simulación de circuitos cuánticos, tanto en un entorno de aprendizaje y experimentación como en investigación científica.

También existen lenguajes diseñados específicamente para la programación cuántica como `QASM`. Además de ofrecer este lenguaje, Quantum Inspire proporciona tanto backends de emulación de circuitos como ejecución por hardware.

Siguiendo en esta línea, `qibo` es un framework desarrollado por el Barcelona Supercomputing Centre (BSC) que comparte similitudes con Python. Ofrece numerosos backends, pudiéndose ejecutar tanto en CPU como en GPU, centrándose a su vez en su facilidad de uso. Además, también ofrecen la ejecución de trabajos en su chip cuántico formado por 5 qubits transmon, basados en superconductividad.

Por otro lado, el CTIC también proporciona un simulador de computación cuántica (QUTE) desplegada en la infraestructura de supercomputación ISAAC y permite la simulación de hasta 38 qubits.

Otra alternativa aparece con `Matlab`, que ofrece soporte para la creación de circuitos y algoritmos cuánticos, con la opción de ejecutar estos programas tanto de manera local como en la nube, utilizando los servicios de AWS o IBM Qiskit Runtime.

Cirq, desarrollado por Google, es otro framework ampliamente utilizado para la simulación y ejecución de algoritmos cuánticos en diversas plataformas, incluidas las de Google Quantum AI. Rigetti Forest es otra alternativa destacada, que incluye simuladores cuánticos y acceso a hardware a través de la nube.

Microsoft ha desarrollado LIQUi|>, un conjunto de herramientas y lenguaje de programación que permite simular algoritmos cuánticos y optimizar circuitos, útil para investigaciones como la teletransportación cuántica. QuEST es otro simulador cuántico de código abierto diseñado para entornos paralelos, permitiendo la simulación eficiente de sistemas cuánticos a gran escala, mientras que ProjectQ, otro marco de código abierto, es utilizado para implementar y ejecutar algoritmos cuánticos en diferentes backends. Finalmente, el Quantum Development Kit de Microsoft y myQLM de Atos son plataformas que permiten

[1] Framework genérico para la búsqueda automática de modelos, no necesariamente cuánticos.

a los usuarios experimentar con algoritmos cuánticos y realizar simulaciones en el campo del QML.

Capítulo 3

# Metodología

## 3.1. Conceptos previos

La llegada de la computación cuántica ha supuesto una gran motivación para el continuo desarrollo de algoritmos cuánticos y arquitecturas que puedan aprovechar todas las características de este paradigma. Sin embargo, este nuevo tipo de computación presenta importantes diferencias con respecto a la computación tradicional que es necesario tener en cuenta:

**Bit vs Qubit:** La primera de estas diferencias la podemos encontrar en la unidad básica de la información. Mientras que el bit solo puede tomar los valores entre 0 y 1, un qubit (acrónimo de quantum bit) puede tomar cualquier superposición[1] entre los estados $|0\rangle$ y $|1\rangle$. Esto le da al qubit una mayor expresividad, pudiendo representar infinitos estados frente a los 2 que posee un bit ordinario. La figura 3.1 representa esta dicotomía.

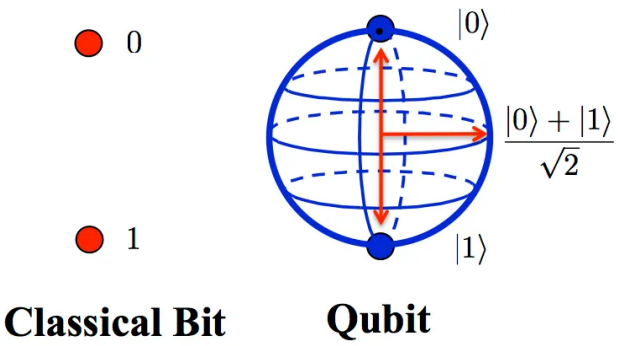


Figura 3.1: Bit (izquierda) y Qubit (derecha).

Por otro lado, los bits son totalmente independientes entre sí. Esto es, se pueden realizar operaciones a cada uno de ellos sin que tenga un efecto en el valor del otro. En cambio, un sistema de varios qubits pueden tener entrelazamiento entre ellos. Esto significa que al realizar una medición en uno de ellos (es decir, colapsar el estado a $|0\rangle$ o $|1\rangle$), el estado en el otro puede verse afectado a pesar de no tener una influencia directa en él.

[1] Los estados cuánticos básicos se representan con la notación de Dirac (bra-ket) como $|0\rangle$ y $|1\rangle$. Estos estados también se denominan estados de la base computacional

**Capa de embedding, encoding o codificación:** Otra importante diferencia derivada del punto anterior es la necesidad de codificar los datos obtenidos de manera clásica para ser interpretados como un estado cuántico. Es por ello por lo que estos modelos necesitan de una capa de embedding o capa de codificación. Se trata de un conjunto de puertas lógicas parametrizables que se utiliza para inicializar un estado en el circuito cuyos valores estarán determinados por los propios datos que se vayan a codificar, traduciendo en ese sentido la información de entrenamiento al mundo cuántico. Matemáticamente, un embedding es una función

$$V : \mathbb{R}^N \longrightarrow \mathcal{H}$$

que asocia el espacio de los datos (visto como $\mathbb{R}^N$ donde $N$ es su número de características) con un espacio de estados $\mathcal{H}$ (espacio de Hilbert) donde $N = 2^n$, siendo $n$ el número de qubits disponible en el circuito. Una buena elección de esta capa permite aprovechar al máximo la dimensionalidad del espacio que ofrece el sistema de qubits. Algunas de las principales codificaciones son el angle embedding, ZZ embedding o amplitude embedding, cada uno con distintas propiedades y características.

**Medición:** Por la misma razón que en el punto anterior, también es necesario transformar el estado cuántico final de un circuito en un resultado que pueda ser interpretado clásicamente en función de la tarea que se pretenda realizar. Para ello, es necesario definir un observable[2] sobre el que realizar la medición y determinar el estado de la base que le corresponda. Utilizando $M = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$ como observable y una vez se colapsa el qubit, de manera natural se puede interpretar observar el estado $|0\rangle$ como 0 y $|1\rangle$ como 1, indicando la pertenencia del input a una clase u otra. En función del número de qubits que se quieran medir, el resultado se puede usar para la clasificación binaria, multiclase o incluso multietiqueta. En este trabajo, el estudio de los modelos se ha centrado en la realización de clasificación binaria.

Otro punto a tener en cuenta a la hora de ejecutar circuitos en hardware cuántico real, es la naturaleza no determinista de la medición, lo que significa que la ejecución del modelo con los mismos datos y parámetros puede dar lugar a resultados totalmente distintos. Por tanto, el resultado de un modelo es una distribución de probabilidad y no un valor fijo determinado.

En lo que sigue en este trabajo, se ha considerado conveniente trabajar con emulaciones ideales porque al trabajar de manera clásica con variables y tipos tradicionales, es posible determinar el estado con mayor probabilidad de aparición como output del modelo sin necesidad de observarlo como estado cuántico. En cambio, al ejecutar circuitos en ordenadores cuánticos, al medir se pierde información sobre el estado en el que se encontraba, con lo que es necesario la ejecución reiterada para conseguir un resultado probabilístico de la salida del modelo.

Por otro lado, también es importante realizar una comparación entre los modelos clásicos y los cuánticos, con el objetivo de estudiar la viabilidad que podrían tener este tipo de estrategias en un futuro próximo en el que se pueda aprovechar todo el potencial que pueden ofrecer los ordenadores cuánticos.

Aun así, debido a encontrarnos todavía en la era NISQ, estas comparaciones no son completamente adecuadas por distintas razones que van más allá del alcance de este trabajo:

[2]En el contexto de la física y la computación cuántica, un observable es una matriz hermítica. Para entender con más detalle sobre los observables y su papel en la computación cuántica, se puede leer en [14]

- No se pueden hacer comparaciones con relación a la potencia, energía gastada, eficiencia energética, intensidad aritmética o tiempos de ejecución (entre otros) si lo que se pretende es comprobar o comparar la efectividad de los modelos. Esto es debido a que a día de hoy, las ejecuciones de los circuitos cuánticos se suelen realizar por medio de emulaciones, con lo que existen numerosos overheads y obstáculos que incrementan el uso de estos recursos y no dependen intrínsecamente del modelo. Por la propia naturaleza de la simulación de qubits, se necesita una cantidad de memoria que crece de manera exponencial. Por otro lado, la efectividad de los modelos se ve fuertemente afectada por el lento desarrollo de los ordenadores cuánticos, que todavía presentan problemas con el ruido cuántico y la decoherencia de los qubits.
- Existen dificultades con la construcción de arquitecturas que puedan ser comparables utilizando el número de parámetros como métrica. Si se quiere construir una CNN con el orden de parámetros similar a las QCNNs aquí presentadas, la red es simplemente demasiado pequeña como para que sea eficaz y pueda aprender, ya que las características se relacionan únicamente con sus vecinas. En el caso cuántico, el estado relaciona todas las características entre sí, haciendo que, el entrelazado de qubits provoque que una puerta cuántica afecte a más qubits (y, por tanto, a más información) que en los que actúa directamente.

  Por otro lado, construir circuitos cuánticos con un número de parámetros similares a las CNNs, mientras que es una buena estrategia para evitar el problema de las *barren plateaus* [4], su ejecución resulta infactible principalmente por los tiempos de entrenamiento necesarios, de nuevo, debido a la emulación de circuitos. Es por esto por lo que se ha intentado buscar arquitecturas en las que se obtuviesen resultados similares entre ellas. De esta manera se puede analizar la efectividad de los modelos respecto al número de parámetros.
- En la actualidad, no se puede determinar la superioridad o no del QML con respecto al ML tradicional, ya que es un campo que todavía se encuentra en fases muy tempranas en su desarrollo, tanto en algoritmia y redes cuánticas como en hardware. Hasta que no lleguemos al fin de la era NISQ, no será posible realizar una comparación fiel entre este tipo de estrategias de inteligencia artificial y Machine Learning. Como ya se ha mencionado anteriormente, un ejemplo de esto es el incremento exponencial de la memoria necesaria, lo que limita enormemente el número de qubits disponibles para realizar emulaciones.

## 3.2. Metodología general

Las aportaciones en este trabajo se pueden distinguir en 3 partes. El primero de ellos, realizar un testeo y benchmark de circuitos cuánticos y arquitecturas de QCNNs. En segundo lugar, comprobar la influencia de la cantidad de muestras y características en la efectividad en los modelos. En última instancia, se lleva a cabo una comparativa entre CNNs y QCNNs, teniendo muy en cuenta la complejidad del modelo y la cantidad de parámetros entrenables. Estos objetivos se verán reflejados en el resto de secciones de este capítulo y el capítulo 4. La figura 3.2 representa un diagrama de flujo con los procedimientos realizados en este documento.

En primer lugar, se ha realizado la preparación del entorno de trabajo. Esto incluye la elección de frameworks de desarrollo de circuitos, selección de dispositivos de simulación y librerías necesarias para el entrenamiento y recopilación de métricas. Todas estas casuísticas se desarrollan en la sección 3.3.

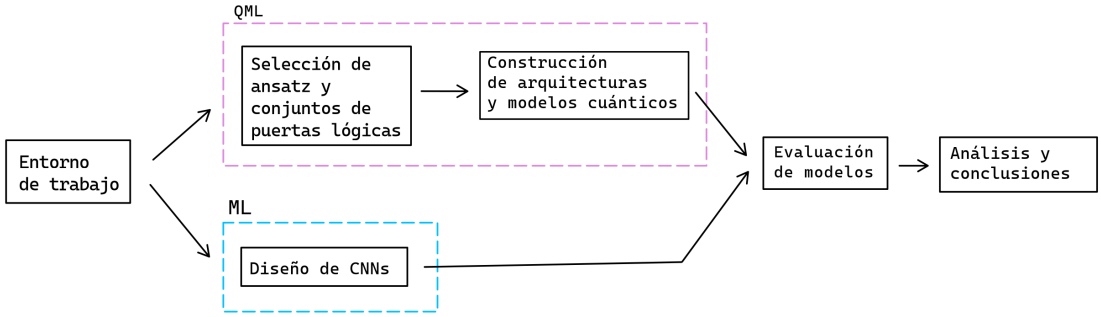


Figura 3.2: Diagrama de flujo de la metodología seguida en este trabajo.

Posteriormente, se realizó la propuesta y selección de circuitos cuánticos y conjuntos de puertas lógicas (también comúnmente denominados como *ansatz*[3]), junto con el diseño de la arquitectura y modelos cuánticos. Durante estas fases iniciales, también se fueron definiendo los tipos de entrenamiento y sus hiperparámetros, como el optimizador a utilizar, el *learning rate* o número de épocas o iteraciones. Todos estos detalles se explican con mayor detalle en la sección 3.5.

Una vez se llevó a cabo la selección y construcción de estos modelos, se efectuó una evaluación más exhaustiva de los modelos cuánticos propuestos. Estos experimentos se detallan en la sección 4.1 del capítulo 4.

Acabado con los modelos cuánticos, tuvo lugar la elaboración de modelos de CNN tradicionales, la contrapartida clásica de las QCNNs estudiadas anteriormente. Con el entorno de trabajo ya establecido, se realizó un sondeo de varios hiperparámetros de la CNN, intentando obtener unos resultados comparables con las redes cuánticas mientras se minimizaba el número de parámetros entrenables de la red. La construcción de CNNs se expone en la sección 3.5.2 y la elección de hiperparámetros en la sección 4.1.

Finalmente, se presentan los resultados obtenidos con mapas de calor y diversas tablas, junto con sus análisis y observaciones concluidas a partir de dichos resultados. La metodología seguida para la extracción de estas conclusiones se expone en la sección 4.2.

El resto de este capítulo está organizado de la siguiente manera. En primer lugar, se mostrarán los frameworks y librerías usadas, seguidas de la metodología que se ha seguido para la elaboración del código y los experimentos. Posteriormente, se contará con mayor detalle todo lo relacionado con los datos y su preprocesado, como el tipo de dataset o como se ha transformado la información disponible. Por último, se explicarán y describirán todos los ansatz y circuitos utilizados en los diversos experimentos, seguido del procedimiento y parámetros del entrenamiento para los modelos presentados aquí.

## 3.3. Frameworks, librerías y estructura del código

Antes de empezar a entrenar modelos y ejecutar circuitos, es muy importante elegir un buen entorno de trabajo y herramientas que nos ayuden a llevar a cabo con éxito los objetivos mencionados con anterioridad. Mientras que se están desarrollando todavía algunos lenguajes y frameworks especializados y orientados específicamente para la construcción y ejecución de circuitos, todavía se encuentran en fases muy tempranas de desarrollo. Es por

[3]Palabra de origen alemán que hace referencia a una solución estimada a una ecuación que describe un problema físico o matemático. En este contexto, se utiliza para hacer alusión a un conjunto diferenciado de puertas lógicas en un circuito cuántico.

ello por lo que se ha decidido utilizar Python, debido a su versatilidad, su rápida curva de aprendizaje, variedad de opciones y soporte de las librerías, documentación y uso generalmente extendido por la comunidad científica para la realización de tareas relacionadas con el Machine Learning.

Dentro de este ecosistema ha sido necesario realizar un sondeo preliminar, probando los principales paquetes de creación de circuitos y de entrenamiento de modelos, buscando una alternativa que ofrezca facilidad y sencillez de uso, pero a su vez con suficiente potencia para poder realizar todas las tareas de QML que aquí nos atañen. A continuación se describirán los principales paquetes y herramientas de las que se ha hecho uso en este trabajo, junto con algunas razones y ventajas de su elección frente a otras alternativas:

`pennylane`: Junto con `qiskit`, se trata de una de las principales librerías de Python para la creación y construcción de circuitos cuánticos. Incluye con numerosos dispositivos de emulación cuántica y backends (`default.qubit`, `lightning.qubit`,...) y gran versatilidad para diseño de circuitos. Se optó por este framework debido a su facilidad de uso y potencia para el entrenamiento de modelos y PQCs, además de contar con gran variedad de recursos y afinidad con otros frameworks y herramientas de entrenamiento y Machine Learning, como pueden ser `jax`, `tensorflow`, `keras` o `torch`.

`jax`: Librería para la aceleración y ejecución de circuitos y cálculos de gradiente, junto con la opción de realizar la ejecución de código (tanto circuitos como iteraciones de entrenamiento) en GPU y otros dispositivos. Incluye su propia implementación de `numpy` y utiliza compilación *just-in-time* para mejorar y optimizar tiempos de ejecución, transformando código de Python en código compilado en C. Sin embargo, el soporte para versiones antiguas y versiones de `cuda` y `jaxlib` hacen que realizar offloading a otros dispositivos y aceleradores sea una tarea con muchas dificultades, por lo que todos los experimentos aquí descritos se han realizado en CPU. Es también compatible con `optax`, otro módulo de Python destinado a la optimización y cálculo de gradientes.

`torch`: Utilizado principalmente para el diseño y entrenamiento de CNNs clásicas. Aunque este paquete también es compatible con `pennylane` para el entrenamiento de circuitos cuánticos, `jax` proporciona tiempos de entrenamiento más rápidos para este tipo de tareas.

`hierarqcal`: Librería para la construcción de jerarquías y arquitecturas de modelos cuánticos. Ofrece gran versatilidad y es compatible con los principales backends de desarrollo de circuitos como `pennylane` y `qiskit`. Permite el diseño de grandes modelos de QML con reducidas líneas de código, centrándose en unas pocas directivas que ofrecen gran expresividad y modularidad, basándose en jerarquías de circuitos y bloques de puertas lógicas.

`librosa`: Paquete de Python utilizado para el análisis de audio y música. Contiene funciones para la extracción de espectrogramas de Mel, MFCCs, STFTs, cambio de escalas,...

`seaborn`: Librería basada en `matplotlib` destinada principalmente para la creación de gráficos, mapas de calor y matrices de confusión, ampliamente usada en labores y experimentos en el campo del Machine Learning.

`wandb`: Librería de Python y herramienta en la nube[4] para el registro y cálculo de métricas

[4] https://wandb.ai/site

en cada iteración del entrenamiento. De esta manera se puede ver automáticamente con gráficas la tendencia de la función de perdida y el accuracy de los conjuntos de train y test a lo largo del experimento, además de ofrecer una buena organización y mantener documentados todas las ejecuciones realizadas. Gracias a esta herramienta, se han podido obtener de manera automática las gráficas, las medias y las desviaciones típicas de cada grupo de experimentos, así como su tiempo de ejecución.

Se considera importante señalar también que tanto `jax` como `torch` y algunos dispositivos de simulación de circuitos de `pennylane` (como los que proporciona la librería `lightning`[5]) utilizan código compilado en C y C++, con lo que, aunque a priori pueda preocupar los tiempos de ejecución y la velocidad de Python, solo se incurre en un pequeño overhead en el tiempo global de ejecución.

Por último, se estima conveniente realizar un pequeño comentario sobre `Catalyst`. `Catalyst` es una librería de Python desarrollado por el mismo equipo que se encuentra detrás de `pennylane` y cuyo objetivo es similar al de `jax`: realizar la compilación de código para acelerar la emulación de circuitos y ejecución en CPU, GPU e incluso QPU. Sin embargo, al estar todavía en fases experimentales, presenta algunas incompatibilidades[6] imposibles de conciliar con el resto de librerías.

Una vez descrito el lenguaje y las principales librerías de las que se harán uso, se expone a continuación la estructura del entorno de trabajo y se explica la función general de los diferentes ficheros que lo componen. Todos estos archivos y procedimientos, junto con la lista de librerías del entorno, se encuentran en un repositorio público alojado en `github`[7]. La figura 3.3 representa el árbol de directorios del entorno, incluyendo tanto los códigos de circuitos y entrenamiento como la estructura de los datasets y las imágenes preprocesadas:

`architecture.py`: Aquí se definen todas las funciones y características relativas a los circuitos o su arquitectura, incluyendo todos los ansatzs, funciones parametrizadas para generar las capas de puertas lógicas y sus jerarquías. también incluyen funciones que representan estos modelos gráficamente, utilizados para mostrar los distintos circuitos que se utilizaran a continuación.

`loader.py`: Aquí se incluyen todas las funciones para el preprocesado del dataset y funciones de carga de los datos, además de la división en conjuntos de entrenamiento y test según los parámetros del experimento a realizar.

`logger.py`: Recopila funciones para el registro de las ejecuciones en `wandb` y la presentación de los resultados en mapas de calor con `seaborn`.

`metrics.ipynb`: Cuaderno de `jupyter` destinado al tratamiento y análisis de los experimentos y resultados, con el fin de representar esta información de manera eficaz y extraer conclusiones de los mismos.

`pennylane_jax.py`: Aquí se establecen todas las funciones relacionadas con un entrenamiento de circuitos cuánticos y QCNNs, definiendo el modelo a partir de las funciones de `arquitecture.py`, junto con el backend y la interfaz del dispositivo. De esta manera, se puede parametrizar de manera más versátil cada tipo de ejecución o entrenamiento, definiendo una función (i.e. `run_jax(...)`) que luego será importada por

[5] https://docs.pennylane.ai/projects/lightning/en/stable/index.html

[6] https://discuss.pennylane.ai/t/catalyst-compilation-failure-operand-1-does-not-dominate-this-use/4239

[7] https://github.com/alejleal/hierarchical-qml-TFM

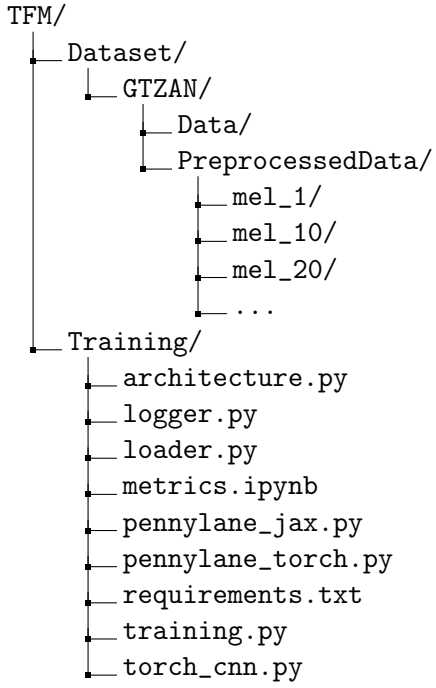


Figura 3.3: Árbol de directorios del entorno de trabajo. Únicamente la carpeta de `Training/` forma parte del repositorio para el control de versiones.

`training.py`, permitiendo una organización más limpia del código. Cualquier otro tipo de entrenamiento o estrategia de ML o QML puede tener una estructura similar a este archivo en el caso en el que se desee implementar otro tipo de estrategias.

`torch_cnn.py`: Script similar al anterior que contiene tanto los modelos clásicos convolucionales como funciones para su entrenamiento y registro de resultados.

`training.py`: Por último, en esta parte del código se establece todo el pipeline e hiperparámetros de cada experimento, como el número de ejecuciones, épocas y parámetros específicos para cada tipo de entrenamiento que se haya definido. Se trata del archivo principal que realiza la llamada al resto de funciones para definir estos experimentos, realizar los entrenamientos y obtener, recopilar y guardar resultados.

## 3.4. Datasets y preprocesado

La elección de los datasets, la calidad de los datos y su preprocesado son tareas cruciales a la hora de realizar un buen entrenamiento y obtener modelos fiables y eficaces. también usado en uno de los experimentos en [8], `GTZAN`[8] es un dataset de pequeña escala comúnmente considerado como el equivalente en audio al dataset de `MNIST`. Este conjunto de datos es ideal para la construcción y testeo de métodos de extracción de características, permitiendo el estudio de los datos desde distintos puntos de vista, tanto como series temporales como en imágenes, como espectrogramas y cromogramas. Los datos que ofrece este dataset son los siguientes:

- 1000 archivos de audio de 10 géneros musicales. Cada género contiene 100 de estos audios con una duración de 30 segundos por cada fragmento de canción.

[8] https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification

- Espectrogramas de Mel de los audios del punto anterior.
- 2 archivos `.csv` que contienen características de cada archivo de audio, como la duración, armonías, frecuencias cromáticas, coeficientes cepstrales de Mel, etc. Uno por la totalidad de cada audio y otro de los 3 primeros segundos y, por tanto, aumentando en 10 la cantidad de datos que se ofrecen en este caso.

No obstante, estas imágenes que se proporcionan no son adecuadas para el entrenamiento. Por un lado, todas las imágenes presentan un marco blanco que no forma parte del espectrograma y, por tanto, empaña la calidad de los datos. Por otro, se ha usado una escala de colores para representar los valores del espectrograma, por lo que nuevamente se está alterando la información que presenta cada muestra del audio, al codificar un único canal de información a 3 (RGB). La siguiente figura (3.4) presenta un ejemplo de esta transformación de señal auditiva a espectrograma:

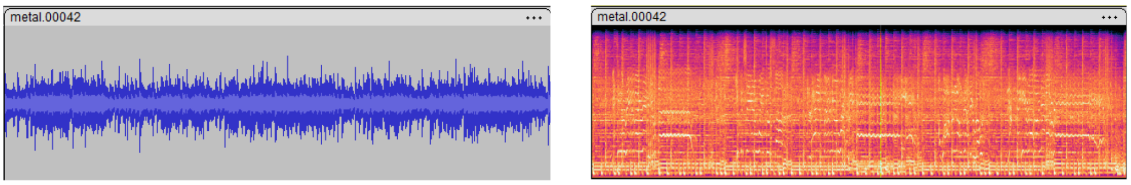


Figura 3.4: A la izquierda, representación como onda del audio. A la derecha, su correspondiente espectrograma de Mel sin aplicar la reducción de características.

Con esta elección del dataset y antes de empezar el entrenamiento, es necesario preparar los datos y extraer características de estos que puedan ser aprendidas e identificar patrones por el modelo. Cuanto mejor se haga esta extracción de características (es decir, puedan ser separadas en el espacio latente y permitan representar e identificar a los datos) más fácil será para el modelo realizar las labores de entrenamiento y obtener buenas predicciones. Por ello, se han realizado distintos tipos de preprocesado en función del número de características y el número de muestras que se extraen de cada audio.

Teniendo como referencia de nuevo a [8], se extraen espectrogramas de Mel de muestras de 3 segundos de cada audio, que servirán como input a los modelos aquí construidos. Esta extracción de muestras se realiza de la siguiente manera en función del número de ellas que se quieran obtener. Sea $n$ este número de muestras:

- Si $1 \leq n \leq 10$, las muestras se obtienen de manera secuencial sin solapamiento. Es decir, se toman los primeros $3n$ segundos del audio divididos en $n$ partes iguales tal y como representa la figura 3.5. El resto del audio se descarta.

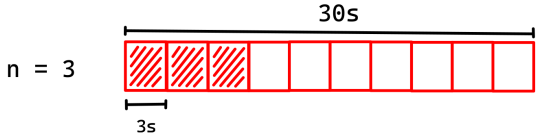


Figura 3.5: Extracción de muestras de un audio para $n = 3$. Los cuadros sombreados corresponden con las muestras que se usarían posteriormente para generar los conjuntos de entrenamiento y test.

- Si $n > 10$, las muestras que se extraen se distribuyen uniformemente a lo largo del fragmento audio, de tal manera que el solapamiento entre dos fragmentos sea el

mismo en todos los casos. Como los audios son de 30 segundos y cada muestra de 3 segundos, para $n$ muestras el solapamiento se puede calcular de la siguiente manera. Si denotamos por $s_n$ a este solapamiento (en segundos):

$$(n-1)\cdot s_n + 30 = 3n \quad \Longrightarrow \quad s_n = 3\cdot\frac{n-10}{n-1} \text{ segundos}$$

Pero para poder extraer cada muestra, es necesario calcular el tiempo entre el inicio de una y la siguiente, con lo que el desplazamiento entre muestras se calcula como $t = 3 - s_n = \frac{27}{n-1}$ segundos. La figura 3.6 representa este tratamiento de la información:

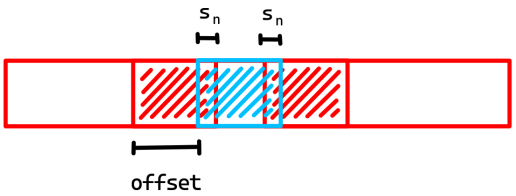


Figura 3.6: Solapamiento entre muestras para $n > 10$.

Una vez divididos los audios, se utiliza la función `melspectrogram` del paquete de análisis de audio de `librosa` con sus parámetros por defecto[9]. Estos parámetros determinan el tamaño de la imagen obtenida después de la transformación. En este caso, esto da lugar a imágenes de tamaño $130 \times 128$ píxeles con un único canal de color. Siguiendo la documentación de la librería, se aplica la función `power_to_db`, con el objetivo de convertir la imagen a una escala que represente mejor las características del espectrograma. El siguiente fragmento de código extrae estos espectrogramas y los guarda en formato `.csv` con el objetivo de facilitar su posterior lectura y tratamiento. Además, mantiene la pureza de los datos, evitando la codificación de esta información como imagen.

```
### loader.py
# load audio as a time series
y, sr = librosa.load(wav_path)

# obtain the spectrogram and rescale it
spec = librosa.feature.melspectrogram(y=y, sr=sr)
spec = np.array(librosa.power_to_db(spec, ref=np.max), dtype='float32')

# save it as a .csv
np.savetxt(f"{path}/{name}.csv", spec, delimiter=',')
```

Como se ha comentado anteriormente, estos valores obtenidos se guardan como un archivo `.csv` (utilizando funcionalidad de `numpy`) para poder realizar la lectura posterior de estos datos de manera más sencilla y evitar preprocesados innecesarios que no estén destinados a la extracción útil de características. En este caso, se separa la obtención de los espectrogramas y la extracción final de características de la imagen con el objetivo de reducir el overhead que surge en la carga del dataset, ofreciendo también mayor versatilidad a la hora de trabajar los datos en un pipeline más modular.

A la hora de realizar los entrenamientos, se leen los valores de estos archivos correspondientes a los géneros que se han elegido y se redimensionan las imágenes utilizando

[9] https://librosa.org/doc/main/generated/librosa.feature.melspectrogram.html

la función `resize` de `tensorflow`, acorde a los parámetros del experimento (como, por ejemplo, las características máximas que se pueden codificar en el sistema cuántico). La siguiente porción de código se ha extraído de la clase `ImageResize` e ilustra la reducción de las imágenes obtenidas anteriormente:

```
### loader.py
def fit(self, X, y=None):
    if self.size == None:
        self.size = X.shape[1] * X.shape[2]
    return self

def transform(self, X, y=None):
    X_resize = tf.image.resize(X[..., np.newaxis][:], (self.size,
    ↪  1)).numpy()
    X_squeezed = tf.squeeze(X_resize).numpy()
    return X_squeezed
```

Posteriormente, esta clase forma parte del pipeline que se encarga de preprocesar los datos e introducirlos al modelo. En el siguiente fragmento de código se puede ver la implementación y lectura de las imágenes correspondiente a una pareja de géneros del conjunto de datos:

```
### loader.py
def dataset_pipeline(ds, pair, nfeat, n, method, train_size):
    # establishes the dataset loading stages and its preprocessing
    pipeline_image = Pipeline([
        ("scaler", ImageResize(size=nfeat))
    ])

    Samples = namedtuple("samples", ["x_train", "x_test", "y_train",
    ↪  "y_test"])

    # function that reads .csv files according to the parameters and
    ↪  returns the data and labels as a tuple
    x, y = get_raw_dataset(ds, pair, method, n)

    # loads the data into different sets for test and training
    image_samples_raw = Samples(*train_test_split(x, y,
    ↪  train_size=train_size))

    # transforms the data according to the pipeline stages
    image_samples_preprocessed = Samples(
        pipeline_image.fit_transform(image_samples_raw.x_train),
        pipeline_image.transform(image_samples_raw.x_test),
        image_samples_raw.y_train,
        image_samples_raw.y_test,
    )

    return image_samples_preprocessed
```

```
### training.py
ds = dataset_pipeline("GTZAN", ["country", "jazz"], nfeat=256, n=10,
↪  method='mel')
```

Esta manera de cargar los datos ofrece modularidad y versatilidad, ya que permite preprocesar los datos de distinta manera o realizar otra transformación determinada con la inclusión de etapas extra al pipeline si en un futuro se requiere.

Toda esta extracción de características variable se realiza con el objetivo de entender con mayor profundidad el efecto de esta selección de la información en el entrenamiento de los modelos. Esto es una parte fundamental del ML, en el que es necesario encontrar un equilibrio entre el preprocesado de los datos y la complejidad del modelo, de tal forma que se pueda llevar a cabo un entrenamiento eficaz, robusto y fiable en función de la tarea que se necesite llevar a cabo.

## 3.5. Modelos y circuitos

En esta sección, se mostrará con mayor detalle el conjunto de modelos, tanto clásicos como cuánticos, que se han puesto a prueba para comprobar la efectividad de estas estrategias de entrenamiento. En primer lugar, se describen los circuitos cuánticos y las QCNNs y posteriormente se dan unas pinceladas sobre la estructura general de la CNN que servirán de comparación para el ML clásico.

### 3.5.1. Redes Quanvolucionales

Antes de explicar con más detalle los circuitos cuánticos propuestos, es preciso describir de manera general de qué partes se compone un modelo de QML. Estos modelos están formados principalmente de 3 etapas o partes: capa de embedding, circuitos o ansatzs y etapa de medición u observación:

- El circuito se inicializa en un primer momento con el estado $|0\rangle$ en todos sus qubits (es decir, el estado $\bigotimes_{i=1}^{n} |0\rangle$ [10]), seguido de la capa de codificación que prepara el estado asociado al input o imagen que se introduce en el modelo.
- La segunda parte está formado por el circuito y sus puertas entrenables; el modelo propiamente dicho.
- Por último, y como se ha mencionado en la introducción del capítulo, también es necesario una etapa de medición para interpretar el estado de manera clásica y determinar el resultado del entrenamiento.

En las siguientes secciones se detallarán con mayor profundidad los circuitos y puertas utilizadas en los modelos de redes quanvolucionales según su orden de ejecución. Para clarificar los siguientes fragmentos de código que se mostrarán a continuación, el paquete `pennylane` para el diseño de circuitos se importa al principio del fichero `architecture.py` de la siguiente manera. Se incluye también algunas de las clases de la librería `hierarqcal` necesarias para estas tareas, ya que aparecerán en numerosas ocasiones a lo largo de esta sección:

```
### architecture.py
import pennylane as qml
from hierarqcal import Qcycle, Qmask, Qinit, Qunitary, Qpermute
```

[10] El símbolo $\bigotimes$ denota el producto tensorial de vectores.

#### 3.5.1.1. Capa de codificación

Debido a la gran cantidad de información con la que se tiene que trabajar, se elige el `AmplitudeEmbedding` como método de codificación, en el que es posible codificar hasta $2^N$ características en $N$ qubits[11], aprovechando de esta manera la capacidad de superposición de estados que la computación cuántica ofrece. La librería `pennylane` ofrece una función que se encarga de llevar a cabo esta tarea:

```
### architecture.py
qml.AmplitudeEmbedding(features=x, wires=range(8), normalize=True,
↪  pad_with=0)
```

Permite la normalización de los datos introducidos, necesarios para la propia codificación del estado en el circuito, junto con la posibilidad de cumplimentar con algún valor las características en el caso de no se hayan completado esas $2^n$ características.

#### 3.5.1.2. Capas de convolución y pooling

Si nos centramos ahora en la arquitectura central de los modelos que se entrena, se ha optado por seguir alguna de las directrices de [8], utilizando una arquitectura también conocida como *tree-tensor*, en el que los qubits disponibles se van reduciendo hasta que solo queda 1, el qubit de medición. Esta arquitectura se construye a partir de las capas de convolución y pooling representadas de la siguiente manera por la figura 3.7:

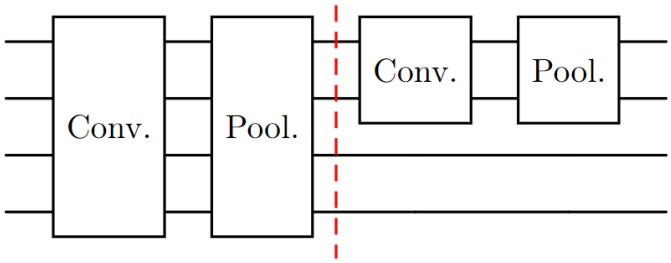


Figura 3.7: Combinación de las capas de convolución y pooling.

Las convoluciones cuánticas se pueden construir a partir de ansatzs de $n \geq 2$ qubits. Estos grupos de puertas lógicas hacen el papel de kernel en la convolución y están distribuidos en los qubits disponibles siguiendo un patrón que permita capturar la esencia de los datos introducidos. En nuestro caso se ha optado por una arquitectura en forma cíclica y cerrada, como se puede ver en la figura 3.8.

Así, se consigue un entrelazamiento entre todos los qubits, introduciendo dependencias entre ellos, lo que resulta vital para aprender y entrenar de manera efectiva. Los parámetros de la capa de convolución se comparten, consiguiendo por un lado reducir la complejidad del modelo y conseguir entrenamientos más rápidos y por otro reducir el efecto de las *barren plateaus* en estos circuitos [9].

Siguiendo el esquema de las QCNNs dadas por [5, 3, 8], las capas de pooling se tratan de capas de entrelazamiento (esto es, capas en las que aparecen puertas controladas) en el que los qubits que actúan de control en la última puerta se dejan de usar en las sucesivas

[11] Si somos más precisos, el número máximo de características que se pueden codificar con este método son $2^N - 1$, ya que es necesario normalizar el estado a norma 1, perdiendo un grado de libertad. Sin embargo, esta pérdida de información no es significativa a medida que el número de qubits aumenta. Se puede leer sobre métodos y estrategias de codificación y embedding en las referencias [2, 11]

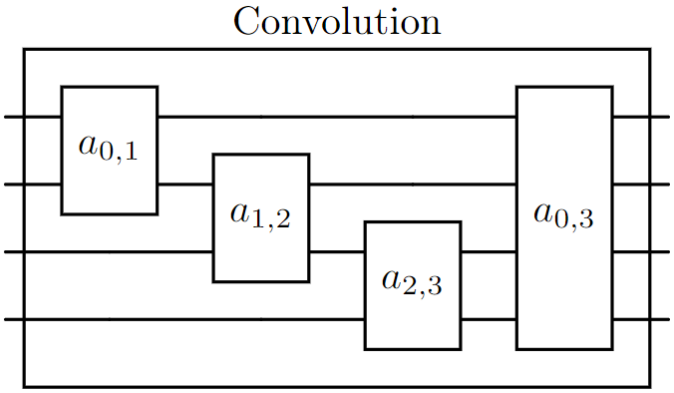


Figura 3.8: Circuito cuántico que realiza la función de convolución en las QCNNs.

capas del modelo. De esta manera, se mimetiza la acción de filtro en estas capas de pooling, como queda representado en la figura 3.9.

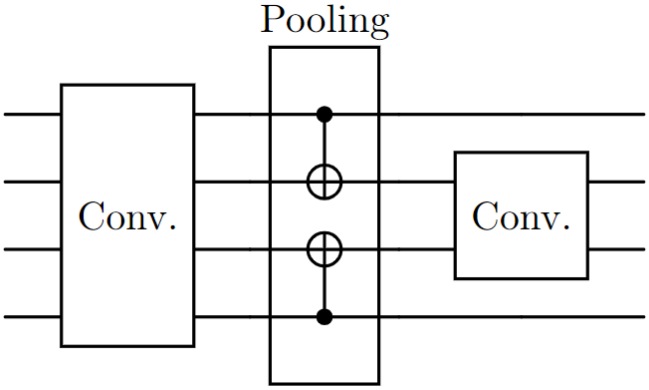


Figura 3.9: Esquema del funcionamiento de las capas de pooling en los circuitos cuánticos usando la puerta *CNOT* como ansatz de pooling.

En este caso, los qubits de los extremos superior e inferior dejan de estar disponibles para futuras capas en el circuito ya que se trata de los qubits de control, mientras que los qubits centrales llevan la información de los datos a las siguientes capas del modelo.

Una vez comentado las partes que conforman una QCNN, se muestra a continuación los principales ansatzs que se han utilizado para el diseño de los modelos puestos a prueba:

- Ansatz $g$: Definido en primer lugar por [5] y después utilizado en [8]. Se ha utilizado para la comprobación de la influencia del número de muestras en los datos, tal y como se ha mencionado y discutido en la sección 3.6. La figura 3.10 muestra la disposición de estos circuitos:

```
### architecture.py
def g(bits, symbols):
    qml.RX(symbols[0], wires=bits[0])
    qml.RX(symbols[1], wires=bits[1])
    qml.RZ(symbols[2], wires=bits[0])
    qml.RZ(symbols[3], wires=bits[1])
    qml.CRZ(symbols[4], wires=[bits[1], bits[0]])
```

```
    qml.CRZ(symbols[5], wires=[bits[0], bits[1]])
    qml.RX(symbols[6], wires=bits[0])
    qml.RX(symbols[7], wires=bits[1])
    qml.RZ(symbols[8], wires=bits[0])
    qml.RZ(symbols[9], wires=bits[1])
```

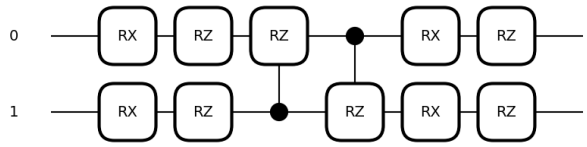


Figura 3.10: Conjunto de puertas que forman el ansatz $g$.

- Ansatz universal ($U_2$): Aunque `hierarqcal` permita el uso de ansatzs de cualquier número de qubits para las capas de convolución, el bloque principal de puertas cuánticas utilizado es el universal de 2 qubits, como se describe en [13]. Esto es, un circuito que es lo suficientemente expresivo para reflejar cualquier estado de 2 qubits a través de sus parámetros y, además, lo hace con un número óptimo de ellos (en este caso, 15).

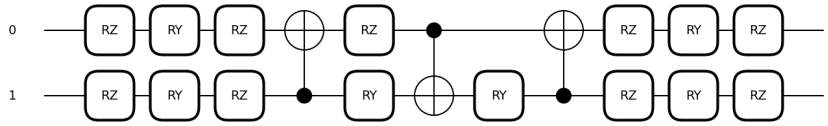


Figura 3.11: Ansatz que puede representar todo estado de 2 qubits con el mínimo número de parámetros.

Las siguientes funciones se encargan de generar el esquema del circuito mostrado en la figura 3.11:

```
### architecture.py
def rot(bits, symbols=None):
    qml.RZ(symbols[0], wires=bits[0])
    qml.RY(symbols[1], wires=bits[0])
    qml.RZ(symbols[2], wires=bits[0])

def universal(bits, symbols=None):
    rot([bits[0]], symbols[:3])
    rot([bits[1]], symbols[3:6])

    qml.CNOT(wires=[bits[1], bits[0]])
    qml.RZ(symbols[6], wires=bits[0])
    qml.RY(symbols[7], wires=bits[1])
    qml.CNOT(wires=[bits[0], bits[1]])
    qml.RY(symbols[8], wires=bits[1])
    qml.CNOT(wires=[bits[1], bits[0]])
```

```
17
18      rot([bits[0]], symbols[9:12])
19      rot([bits[1]], symbols[12:])
```

- Ansatz de pooling (4way): El ansatz más simple para realizar la función de las capas de pooling es simplemente la puerta CNOT. Sin embargo, esto limita mucho la capacidad de entrenamiento del modelo. Utilizando ansatzs mas complejos y con parámetros entrenables pueden aumentar la expresividad del circuito y mejorar los resultados obtenidos.

  La librería `hierarqcal`, con el objetivo de dar una mayor versatilidad y libertad para la creación de circuitos, permite que las capas de pooling también dispongan de parámetros entrenables. El circuito que se ha utilizado para este papel se representa con el esquema de la figura 3.12, junto con la función que genera dicho circuito:

```
1  ### architecture.py
2  def pool(bits, symbols):
3      qml.CRZ(symbols[0], wires=[bits[0], bits[1]])
4      qml.PauliX(wires=bits[0])
5      qml.CRX(symbols[1], wires=[bits[0], bits[1]])
```

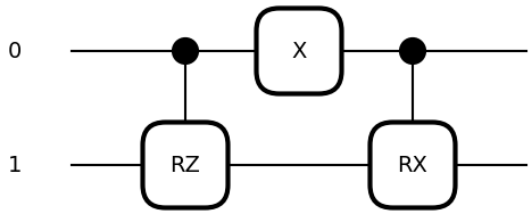


Figura 3.12: Ansatz utilizado para las puertas de pooling.

  Con esta definición del circuito, cada ansatz de pooling presente en el modelo hace que el qubit correspondiente al 1 en cada caso pueda tener hasta 4 transformaciones diferentes, según corresponda la actuación de la puerta $R_z$ o la $R_x$. Suponiendo que en una capa determinada se disponga de $N$ qubits, los estados posibles después de la ejecución de la capa de pooling ascienden a $4 \cdot N/2 = 2N$ estados, lo que le da una gran expresividad a la hora de reducir el número de información disponible para futuras capas. La figura 3.13 muestra un diagrama donde se pueden ver representadas estas 4 posibilidades.

Para construir las QCNNs, se han construido 2 arquitecturas diferentes utilizando la librería `hierarqcal` comentada en la sección 3.3 mostrando la facilidad para definir circuitos. Para el cálculo del número de parámetros en la arquitectura, se supone en todos los casos que las capas de convolución y pooling se comparten los mismos parámetros[12]:

- La primera arquitectura sigue una estructura de árbol binario conocida también con el nombre de tree-tensor [2]. Las capas de convolución están formadas por los

[12] Los de convolución y pooling por separado

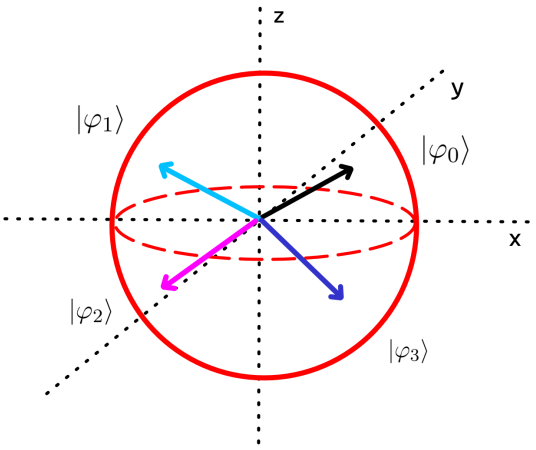


Figura 3.13: Estados alcanzables ($|\varphi_0\rangle$, $|\varphi_1\rangle$, $|\varphi_2\rangle$ y $|\varphi_3\rangle$) por el ansatz en función del qubit de control y el estado inicial $|\varphi_0\rangle$.

ansatz comentados anteriormente, dispuestas en forma de ciclo, determinado por los hiperparámetros stride, step y offset, todos con sus valores por defecto. En cuanto a las capas de filtrado o pooling, estas reducen el número de qubits disponibles a la mitad hasta que quede un solo qubit correspondiente a la salida del modelo. Por tanto, una de las restricciones para seguir esta arquitectura es tener disponibles exactamente un número de qubits igual a una potencia de 2, es decir, $2^k$ qubits. Este número $k$ a su vez se corresponde con el número de capas de convolución y pooling, haciendo que la arquitectura presente $(p_{conv} + p_{pool}) \cdot k$ parámetros en total, donde $p_{conv}$ y $p_{pool}$ es el número de parámetros del ansatz de convolución y el de pooling respectivamente. Esta jerarquía se puede generar con las siguientes líneas de código simplificado, mientras que la figura 3.14 enseña un esquema de esta arquitectura:

```
### architecture.py
# "conv" and "pool" are the ansatzs for the convolution and pooling
↪  respectively, corresponding to the functions defined previously
conv_map = Qunitary(conv, n_symbols=conv_n_symbols, arity=2)
pool_map = Qunitary(pool, n_symbols=pool_n_symbols, arity=2)

# convolution + pooling layer (second qubit as control: "01")
conv_and_pool = Qcycle(mapping=conv_map) +
                Qmask(filter="01", mapping=pool_map)

# initialize qubits + layers * (conv + pool)
qcnn = Qinit(range(wires))
qcnn += ((wires - 1).bit_length() * conv_and_pool)
```

Si quitamos la restricción en el número de qubits, una idea natural para generalizar esta arquitectura consiste en definir la primera capa de pooling de tal manera que deje disponibles un número de qubits equivalente a la mayor potencia de 2 posible. Es decir, si se dispone de $n$ qubits, la primera capa reduce el número de qubits a $2^{\lfloor \log_2 n \rfloor}$. El número de parámetros en este caso es similar al anterior, ascendiendo el total de parámetros a $(p_{conv} + p_{pool}) \cdot (k + 1)$ ya que se necesita de una capa extra. La figura 3.15 muestra como funcionaría esta capa inicial.

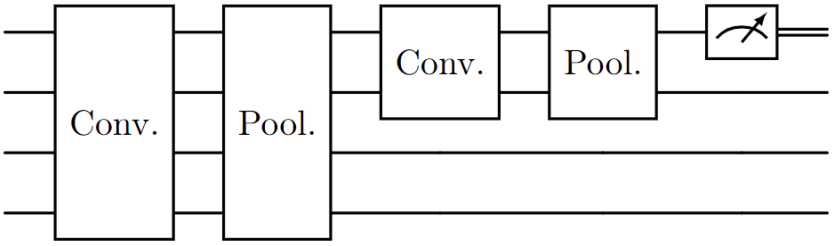


Figura 3.14: Esquema de la arquitectura del circuito *tree-tensor* para 4 qubits.

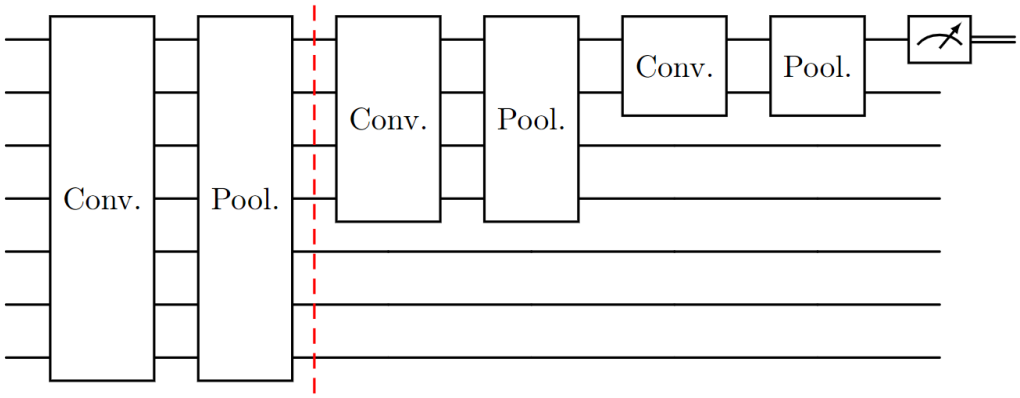


Figura 3.15: Esquema de la arquitectura *tree-tensor* extendido para 7 qubits. La primera capa de pooling hace que los 3 últimos qubits dejen de estar disponibles.

- La segunda de las arquitecturas se focaliza en trabajar con los qubits centrales. En cada capa de pooling, los qubits situados en los extremos dejan de estar disponibles para la siguiente capa de convolución, reduciendo el número de qubits disponible en 2. En este caso, el número de capas crece de manera lineal con el número de qubits. Si se dispone de $n$ qubits, el número de capas se corresponde con $\lfloor \frac{n}{2} \rfloor$. En el caso de que $n$ sea par, la última capa de pooling toma el primero de ellos como qubit para la medición final. El número de parámetros de esta jerarquía es de $(p_{conv} + p_{pool}) \cdot \lfloor \frac{n}{2} \rfloor$. La figura 3.16 representa el diseño de esta arquitectura.

```
### archtecture.py
# convolution + pooling layer (edge qubits as control: "1*1")
conv_and_pool = Qcycle(mapping=conv_map) +
                Qmask(filter="1*1", mapping=pool_map)

# initialize qubits + layers * (conv + pool)
qcnn = Qinit(range(wires))
qcnn += ((wires//2) * conv_and_pool))
```

#### 3.5.1.3. Capa de medición

Una vez descritas la estructura de los modelos que se van a contemplar, para construir el circuito final es necesario incluir la capa de embedding y la función de medición. Para devolver los resultados del modelo, se ha utilizado la función `pennylane.expval`, que calcula el valor esperado por un observable dado. En nuestro caso, si nuestro observable es

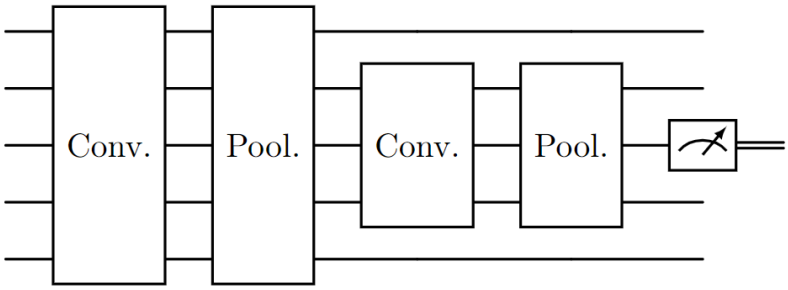


Figura 3.16: Esquema de la arquitectura "central" para 5 qubits.

$M = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$, el valor retornado por el circuito corresponde con la probabilidad de observar el estado $|0\rangle$ en el qubit final. La siguiente función extraída de `architecture.py` se encarga de devolver el circuito o modelo a entrenar dada la jerarquía a usar, el dispositivo que se usará para la simulación y la interfaz con la que se ejecuta el entrenamiento (`jax` o `torch`):

```
### architecture.py
def get_circuit(qcnn, device, interface):
    dev = qml.device(device, wires=qcnn.tail.Q)

    @qml.qnode(dev, interface=interface)
    def circuit(x, symbols):
        # set parameters in the model
        qcnn.set_symbols(symbols)

        # embedding
        qml.AmplitudeEmbedding(features=x, wires=qcnn.tail.Q,
        ↪  normalize=True, pad_with=0)

        # qml model
        qcnn(backend="pennylane")

        # measurement
        M = [[1, 0], [0, 0]]
        return qml.expval(qml.Hermitian(M, wires = qcnn.head.Q[0]))
return circuit
```

### 3.5.2. Redes Convolucionales

Cambiando nuestra atención sobre modelos clásicos, las CNNs se concibieron como resultado de la necesidad para tratar con una gran cantidad de información en los datos pero a su vez manteniendo una complejidad del modelo manejable, sin incurrir en un aumento incontrolable en el número de parámetros. Esto se consigue por medio de lo que se denomina como convoluciones, en las que los coeficientes de los kernels son iguales para el tratamiento de todas las características. En nuestro caso, se han considerado CNNs siguiendo una estructura VGG sencilla y con poca profundidad, con el objetivo de hacer una comparación con una implementación cuántica similar a esta. Se ha llevado a cabo un primer estudio de la estructura de estas CNNs para reducir al máximo posible el número de parámetros de la red, intentando mantener resultados comparables a los obtenidos con

QCNNs. Estas CNNs están formadas por:

- 2 convoluciones con número parametrizable de filtros y tamaño de kernel. Entre estas dos capas de convoluciones se encuentra una capa de MaxPooling que reduce el tamaño de las imágenes a la cuarta parte de su tamaño.
- 2 capas lineales después de estas convoluciones, también con un número de neuronas parametrizables.
- Funciones de activación `ReLU` después de cada una de estas capas, salvo la última de ellas sucedida por una función de `softmax`[13] para normalizar el output y obtener una predicción para la clasificación de la imagen.

```
class Net(nn.Module):
def __init__(self, size, filters, linear_sizes):
    super().__init__()
    k1, k2 = 3, 3       # kernel sizes
    c1, c2 = filters       # convolution filters
    red_dim = lambda x: (x - k1 + 1)//2 - k2 + 1
    in_linear = red_dim(size[0])*red_dim(size[1])
    l1, l2 = linear_sizes

    self.conv1 = nn.Conv2d(1, c1, k1)
    self.pool = nn.MaxPool2d(2, 2)
    self.conv2 = nn.Conv2d(c1, c2, k2)
    self.fc1 = nn.Linear(c2 * in_linear, l1)
    self.fc2 = nn.Linear(l1, l2)
    self.fc3 = nn.Linear(l2, 2)

def forward(self, x):
    x = self.pool(F.relu(self.conv1(x)))
    x = F.relu(self.conv2(x))

    x = torch.flatten(x, 1) # flatten all dimensions except batch
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)

    x = F.softmax(x)
    return x
```

La figura 3.17 representa de manera visual esta red convolucional descrita aquí.

## 3.6. Entrenamiento de los modelos

Siguiendo con lo expuesto en el capítulo 2, existen diferentes tipos de estrategias y entrenamientos dentro del mundo del QML. El tipo de entrenamiento más extendido actualmente para realizar estas tareas consiste en dividir este procedimiento en dos partes: la

[13] La función `softmax`, habitualmente representada por la letra $\sigma$, se utiliza para normalizar un vector de $\mathbb{R}^N$ al hipercubo unidad $[0,1]^N$ con la finalidad de dar una distribución de probabilidad a la salida del modelo.

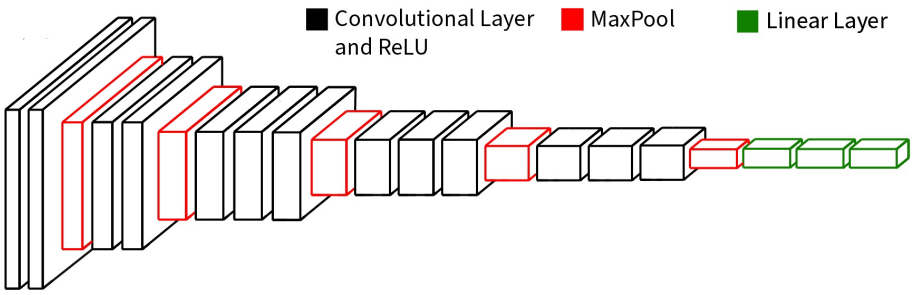


Figura 3.17: Esquema general de la arquitectura de la red convolucional que se utilizará como análogo clásico de las QCNNs. Fuente: [12].

ejecución del circuito cuántico con sus parámetros y la actualización de estos parámetros en un ordenador clásico.

El esquema general del entrenamiento se puede distinguir en las siguientes etapas, independientemente de si se trate de un modelo clásico o cuántico:

1. Preparación de parámetros y variables, como semillas y accuracy acumulado e inicialización de las runs para el registro de métricas y datos de entrenamiento con la herramienta `wandb`.
2. Lectura y preparación del dataset.
3. Ejecución del experimento. Por cada run que se vaya a repetir:
   - 3.1 Inicialización aleatoria de parámetros del modelo.
   - 3.2 Entrenamiento del modelo. Por cada época, se realiza, en este orden:
     - Ejecución del modelo.
     - Cálculo de la función de coste.
     - Actualización de parámetros del modelo.
4. Recopilación y cálculo de resultados y obtención de gráficas.

Con el propósito de realizar una comparativa lo más imparcial posible, se han mantenido todos los hiperparámetros de entrenamiento iguales para todo el abanico de modelos, a excepción del learning rate como se comentará más adelante. Tampoco se ha realizado validación cruzada debido al limitado tamaño de los datos de entrenamiento para el caso de la extracción de una única muestra.

Se ha optado por un tipo de entrenamiento tradicional basado en descenso de gradiente por batch, en el que, para acelerar tiempos de entrenamiento y poder realizar una mayor cantidad de ejecuciones, se utiliza la totalidad del conjunto de entrenamiento. De esta manera, en el caso de los modelos cuánticos, `jax` permite paralelizar la ejecución de los circuitos con cada una de las imágenes del conjunto. La división del dataset se realiza con una proporción de train-test del 70 %/30 %.

En cuanto al optimizador para el cálculo del gradiente en cada iteración, se ha optado por `Adam`. proporcionado por la librería `optax` en el caso de los modelos cuánticos y por `torch` en el caso de las CNN clásicas. Este optimizador se basa en el cálculo de momentos y adapta el learning rate para cada parámetro en función de la magnitud de los gradientes en épocas anteriores. Se trata de una alternativa que generalmente muestra buenos resultados para una amplia variedad de tipos de entrenamiento, especialmente para cantidades grandes

de datos y características. Se ha elegido un learning rate de 0,01 para las QCNNs y uno de 0,001 para las CNNs. Estos valores fueron elegidos durante fases iniciales del desarrollo de toda la estructura del código y testeo de modelos iniciales, obteniendo tendencias de entrenamiento aceptables.

Al tratarse de clasificación binaria, se ha decidido por el uso de *Binary Cross-Entropy* (BCE) como función de pérdida. Si lo unimos con el uso de batch para el entrenamiento, esta función se define como:

$$\mathrm{BCE}(Y, \hat{Y}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

donde $Y$ es el conjunto de etiquetas o clases reales de las imágenes de entrenamiento e $\hat{Y}$ es el de las predicciones realizadas por los modelos. Los elementos de $Y$ solo pueden tener los valores 0 o 1, mientras que las predicciones pueden tomar valores en todo el intervalo [0, 1], ya que denotan el grado de certeza a la pertenencia de una clase o la otra.

Por último, para comprobar la tendencia de la función de pérdida y el accuracy en el conjunto de test, se realiza un experimento simple con la arquitectura de la figura 3.14 con 8 qubits y el ansatz $g$ (figura 3.10). Con respecto al dataset, se toma únicamente con 1 sola muestra por audio y 256 características por imagen, saturando la capacidad de esos 8 qubits. En particular, se eligen las parejas de géneros blues-classical, country-jazz y metal-pop ya que a priori representan el rango de dificultad completo que nos podemos encontrar a la hora de realizar posteriormente ejecuciones del modelo, como se observa en la figura 3 en [8]. Se extiende el entrenamiento por 500 épocas, repitiéndose un total de 10 veces por pareja de géneros, con el objetivo de observar el progreso de la función de pérdida y determinar un buen número de iteraciones para los experimentos futuros. Las siguientes figuras 3.18 y 3.19 muestran el curso de estas 2 métricas, donde el sombreado representa los mínimos y máximos en cada época.

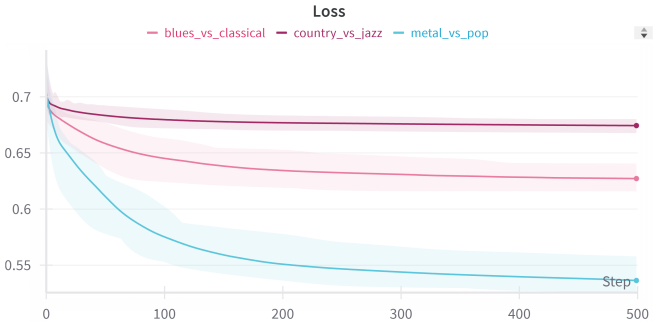


Figura 3.18: Función de pérdida.

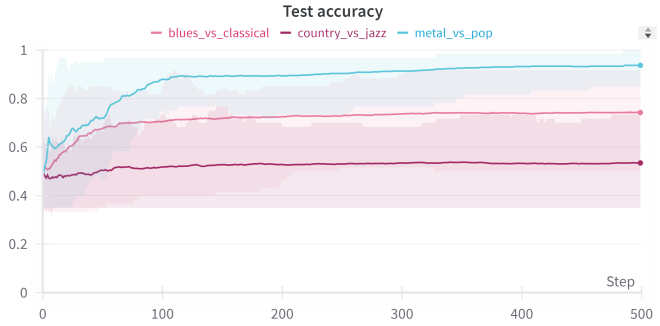


Figura 3.19: Accuracy del modelo.

En la figura 3.18 se puede apreciar como los géneros más difíciles de distinguir (blues-classical y country-jazz) apenas muestran variaciones alrededor de la época 200, observándose este mismo fenómeno en el accuracy del modelo. Para la pareja metal-pop, este descenso es más pronunciado y el aprendizaje necesita de más épocas para estabilizarse. Aun así, la efectividad en este último caso se estanca sobre la época 100, con una precisión del 88 %, e incrementándose muy lentamente hasta el 93 % final.

Es por ello por lo que se decidió elegir 250 como número de iteraciones máximo para cada ejecución: no es demasiado bajo como para finalizar el entrenamiento de manera prematura, pero le da tiempo suficiente para entrenar y estabilizar la función de coste.

Capítulo 4

# Experimentación y resultados

## 4.1. Modelos propuestos y experimentos

Para la experimentación y estudio de los principales modelos y arquitecturas, se ha realizado una clasificación binaria de los géneros de música que ofrece el dataset `GTZAN`, como se ha comentado en la sección 3.4. Al tener disponibles 10 géneros musicales, el número de parejas que se pueden obtener entre ellos son $\binom{10}{2} = 45$ entrenamientos en total por ejecución y experimento. Esto permite sacar conclusiones sobre las diferencias existentes entre géneros, dando información sobre aquellas parejas que más similitudes tienen entre sí y son más difíciles de distinguir a partir de su espectrograma.

En relación con el quinto objetivo comentado en el capítulo 1, se han realizado experimentos en 2 dimensiones diferentes con respecto a los datos: el número de muestras extraídas por audio y número de características por imagen, esto es, la resolución de los datos:

**Número de características:** En cuanto a la elección en el número de características a extraer, esta se basa fundamentalmente en la cantidad máxima de información que es posible codificar en un conjunto de qubits. Para un sistema de $n$ qubits, esto corresponde a un máximo de $2^n$ características por muestra del dataset. La codificación en las amplitudes de los estados puede llevar a cabo esta tarea, como se ha mencionado con anterioridad en la sección 3.5. Teniendo en cuenta que se ha utilizado el circuito de 8 qubits de [8] como referencia y que al añadir 1 qubit al sistema el máximo de características se duplica, se ha decidido utilizar circuitos de 8, 9 y 10 qubits. Esto da lugar a la extracción de 256, 512 y 1024 características por imagen.

**Número de muestras:** Los autores de [8] utilizan únicamente los 3 primeros segundos de cada audio para representar al género completo (tal y como aparece en uno de los preprocesados del propio dataset). De esta manera se restringe en gran medida toda la información presente en los fragmentos de cada canción disponible por género, por lo que para ver la influencia de esta elección en la precisión de los modelos, se ha considerado el aumento de muestras extraídas de cada audio.

Debido a esto y previo a la ejecución de todos los experimentos principales, se realizó una prueba con la extracción de más muestras del dataset. El modelo cuántico utilizado se compone de 8 qubits y presenta la arquitectura *tree-tensor* comentado en la sección 3.5. Los mapas de calor de las figuras 4.1 y 4.1 muestran 2 ejecuciones

utilizando el optimizador e hiperparámetros comentados anteriormente en la sección 3.6 después de 50 épocas. Cada entrenamiento se repite 5 veces.

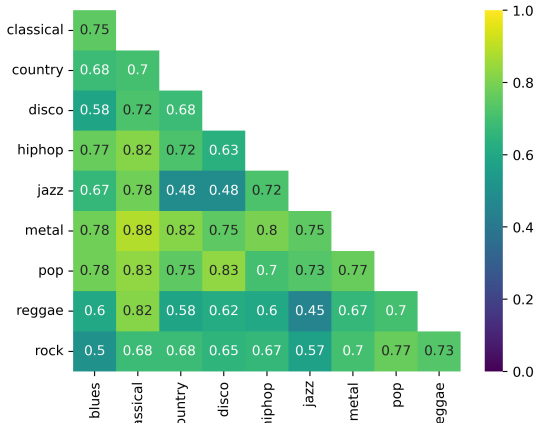


Figura 4.1: Accuracies con la extracción de una (1) muestra por audio.

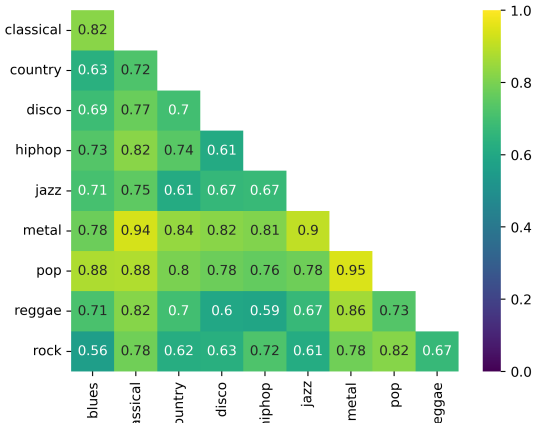


Figura 4.2: Accuracies con la extracción de 10 muestras por audio.

Como se puede observar, la mejoría es notable en todos los enfrentamientos, lo que a priori da validez a la motivación y el estudio de la influencia de la cantidad de datos en el entrenamiento.

Así, los principales modelos se entrenan obteniendo 1, 10 y 20 muestras para cada uno de ellos, dando así lugar a un amplio abanico de posibilidades. Para la variante de 1 muestra, se utilizan únicamente los 3 primeros segundos, como en el experimento original. Esto también permite observar la efectividad de los modelos con poca cantidad de datos de entrenamiento. Las alternativas con 10 y 20 audios hacen uso de cada archivo por completo. En el primero de los casos, cada muestra extraída es adyacente a la anterior y no comparten información entre ellas. Para el caso de las 20 muestras, estos fragmentos sí que presentan solapamiento entre ellas y se distribuyen uniformemente a lo largo del audio.

Esta extracción de muestras permite ejecutar el experimento con 200, 2000 y 4000 imágenes por entrenamiento (esto es, cada pareja de géneros), respectivamente, dando lugar a las siguientes distribuciones de imágenes por pareja de géneros para cada número de muestras extraídas por audio, como se puede observar en la tabla 4.1.

| Número de muestras por audio | 1 | | 10 | | 20 | |
|---|---|---|---|---|---|---|
| | *Train* | *Test* | *Train* | *Test* | *Train* | *Test* |
| Tamaño del conjunto | 140 | 60 | 1400 | 600 | 2800 | 1200 |

Tabla 4.1: Tamaño de los conjuntos de *train* y *test* en función del número de muestras extraídas en cada audio.

El total de imágenes obtenidas para todo el dataset asciende a 1000, 10000 y 20000 ejemplos en cada caso.

De esta manera, se tienen hasta un total de 9 posibilidades con datos con distintas resoluciones y con mayor o menor cantidad de muestras. Con todo este rango de experimentos,

podemos observar la efectividad y la importancia de esta faceta del entrenamiento para los modelos de QML. Estos experimentos se repiten un total de 5 veces por cada par de géneros con el objetivo de dar una visión más general del comportamiento de cada modelo con respecto al entrenamiento.

En cuanto a los modelos que se han testeado en este trabajo, se diferencian entre los circuitos cuánticos y las CNNs:

**QCNN:** Las QCNNs aquí utilizadas tienen una fuerte relación con la cantidad de características elegidas para la resolución de las imágenes. Se presentan 3 modelos de QCNN diferentes en arquitectura, pero utilizando los mismos ansatzs de convolución y pooling en los 3 casos: el universal de 2 qubits (figura 3.11) y el 4way (figura 3.12), respectivamente:

- Para el dataset con 256 características, se ha hecho uso de un modelo con la arquitectura en forma de árbol binario (figura 3.14) y 8 qubits. Esto es debido a que es necesario una potencia de 2 en el número de qubits para poder construir de manera adecuada este tipo de circuitos y además, $2^8 = 256$, con lo que se permite usar la expresividad completa de los estados que presentan esos 8 qubits. Este modelo presenta 51 parámetros entrenables.
- Para el dataset con 512 características, es necesario un modelo con al menos 9 qubits para poder codificar toda la información en los estados cuánticos. Al tratarse de un número impar de qubits, la jerarquía más adecuada a utilizar es la "central", como se mostraba en la figura 3.16. Este circuito tiene 68 parámetros entrenables.
- Por último, para la variante del dataset con 1024 características, se ha construido un modelo de 10 qubits. La arquitectura extendida del árbol binario es adecuada para esta tarea, ya que permite la construcción de circuitos con cualquier número de qubits. Al igual que el anterior modelo, este circuito presenta 68 parámetros entrenables.

**CNN:** Como ya se comentó en el capítulo 3, los modelos de QML y ML no pueden ser totalmente equiparables. Aun así, se ha intentado realizar una comparación que permitiese representar diferencias presentes entre estos tipos de estrategias de aprendizaje automático. Por ello, utilizando la CNN mostrada en la sección 3.5.2, se han explorado algunos de estos modelos modificando el número de filtros y número de neuronas en las capas fully-connected, pero sin pretender ser demasiado exhaustivos en esta búsqueda.

Las figuras 4.3, 4.4 y 4.5 muestran los accuracies sobre todo el dataset con 1 muestra por audio. En los 3 casos se han obtenido resultados que a simple vista son comparables a los vistos en [8] tomados como referencia. A pesar de que a primera vista, el primer modelo (figura 4.3) presenta resultados ligeramente mejores de manera global, el modelo de la figura 4.5 presenta el mínimo número de parámetros entrenables como muestra la tabla 4.2, con lo que se tomará esta CNN para la posterior ejecución del resto de experimentos.

La tabla 4.3 resume y organiza los tipos de modelos usados en cada caso y sobre qué conjunto de datasets se evalúa.

La realización de todos estos experimentos, pruebas y ejecuciones han sido llevadas a cabo en un sistema facilitado por la Universidad de Oviedo, formado por 5 nodos de cálculo específicamente preparados para la realización de tareas de esta índole. La ejecución de

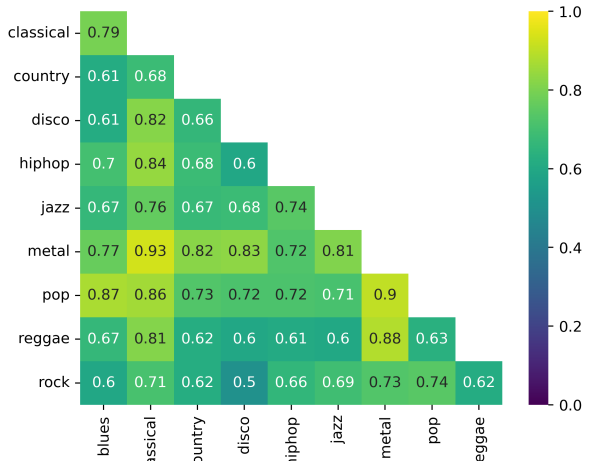


Figura 4.3: Filtros: [3, 7]
FC: [120, 84].

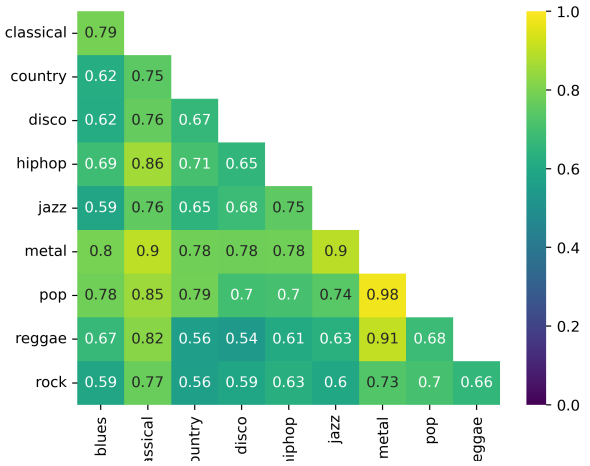


Figura 4.4: Filtros: [3, 7]
FC: [80, 40].

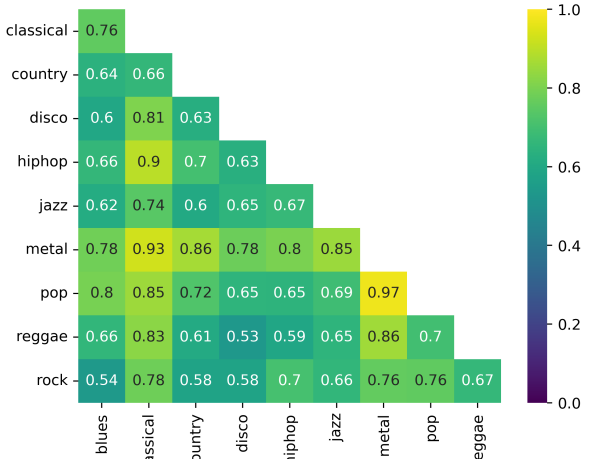


Figura 4.5: Filtros: [3, 5]
FC: [80, 40].

| Nº Filtros<br>Nº Neuronas en FC | [3, 7]<br>[120, 84] | [3, 7]<br>[80, 40] | [3, 5]<br>[80, 40] |
|---|---|---|---|
| Parámetros | 21600 | 10908 | **8772** |

Tabla 4.2: Parámetros entrenables en función de los hiperparámetros de la CNN.

estos entrenamientos ha sido posible gracias a un sistema de colas que administra automáticamente los nodos disponibles y distribuye las tareas según un conjunto de variables establecido como el número de núcleos de CPU a utilizar o GPUs sin carga de trabajo. Es por esto por lo que no se ha considerado tener en cuenta los tiempos de ejecución en el análisis de los resultados, debido a las disparidades entre las prestaciones de cada CPU presentes en los nodos.

| Nº características | 256 | | | 512 | | | 1024 | | |
|---|---|---|---|---|---|---|---|---|---|
| Nº Muestras | 1 | 10 | 20 | 1 | 10 | 20 | 1 | 10 | 20 |
| QCNN | 8 qubits *tree-tensor* $U_2$ + 4way | | | 9 qubits centro $U_2$ + 4way | | | 10 qubits *tree-tensor* $U_2$ + 4way | | |
| CNN | nº filtros: [3, 5] fc: [80, 40] | | | | | | | | |

Tabla 4.3: Resumen con los modelos y sus parámetros (tanto QCNNs como CNNs) para cada tipo de entrenamiento en función del preprocesado del dataset.

## 4.2. Resultados

En esta sección se presentarán los resultados obtenidos de los experimentos descritos anteriormente, junto con una exploración de ellos teniendo en cuenta diversos aspectos y parámetros relacionados con el entrenamiento. Las principales métricas recogidas a lo largo de la ejecución de estos experimentos son las siguientes:

**Accuracy de test:** Este valor describe la precisión del modelo en un conjunto de datos en los que no ha entrenado. Esto es, por tanto, una buena manera de indicar si el modelo ha generalizado bien de los datos de entrenamiento.

**Media de accuracy entre ejecuciones ($\mu$):** Esta métrica va asociada a la repetición de los experimentos para dar validez estadística y determinar el rendimiento general de los modelos probados. El valor que se utiliza para obtener esta media es el del accuracy de test en la última época de entrenamiento.

**Desviación típica ($\sigma$):** Al igual que la media, la desviación típica está asociado a la repetición de **runs**.

**Coeficiente de variación:** Por último, el coeficiente de variación se define como el cociente entre la desviación típica y la media comentado en los puntos anteriores. Se busca que los valores obtenidos en esta métrica sean reducidos, indicando que los datos de la muestra están concentrados con respecto a su media teniendo en cuenta su magnitud.

El análisis realizado de estos resultados se divide en 2 partes diferenciadas por cada tipo de modelo (QCNNs y CNNs). Por un lado, se estudiarán las métricas obtenidas por 3 características del dataset y procesado: por número de muestras extraídas, por resolución de las imágenes (número de features de los datos) y por género musical. Por otro lado, se dará una visión más global de los resultados, atendiendo al coeficiente de variación y dispersión de las accuracies de los 9 experimentos, comentando también si se observa o no overfitting en alguno de estos modelos. Se presentará en primer lugar los resultados para los modelos cuánticos y posteriormente los clásicos.

### 4.2.1. QCNNs

A continuación, en la figura 4.6 se muestran los mapas de calor correspondientes a las accuracies entre pares de géneros en función del número de características y de muestras, como se ha mencionado en los experimentos realizados:

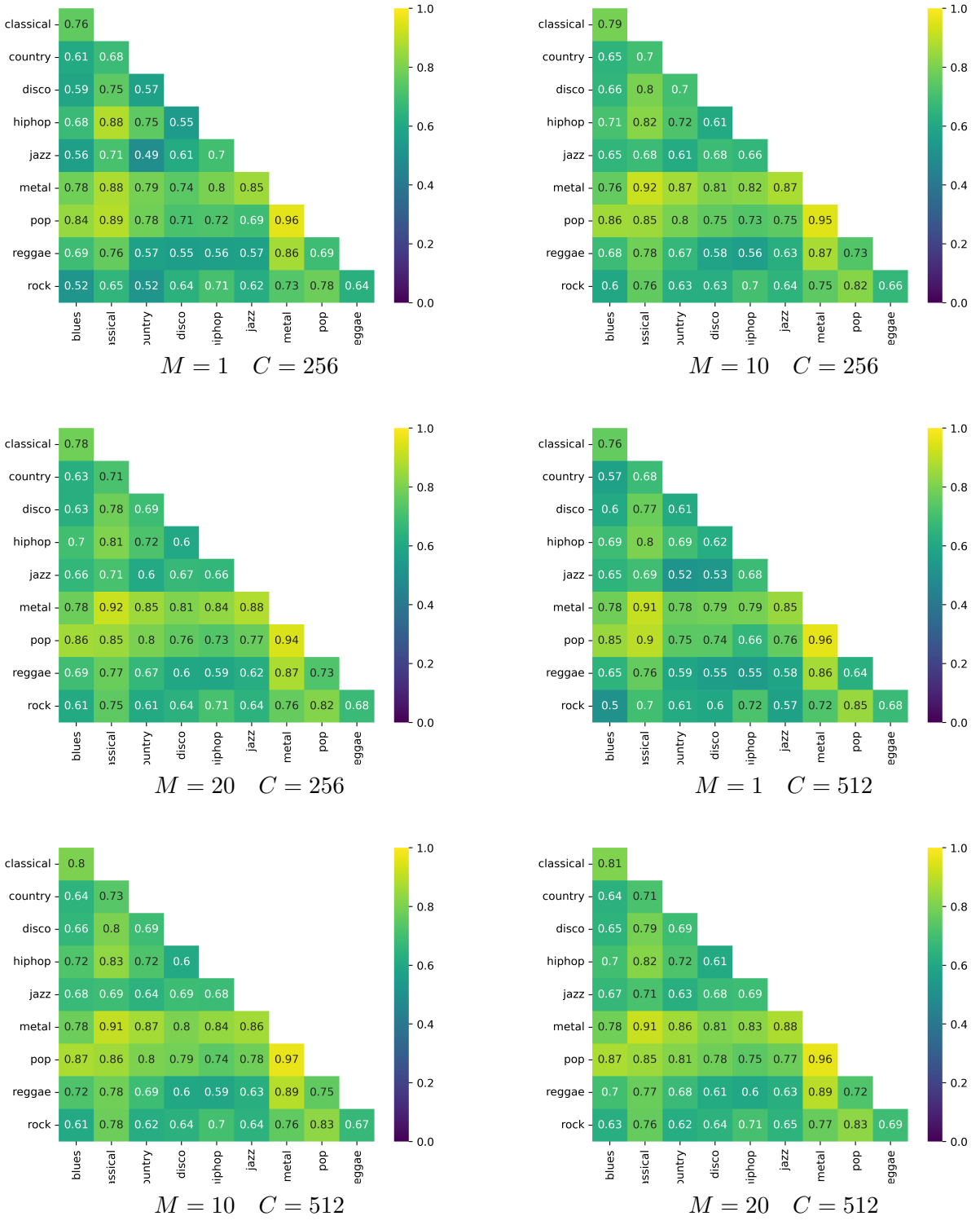


Figura 4.6: Accuracies de los entrenamientos de las QCNNs. $M$ indica el número de muestras y $C$ hace referencia al número de características de cada ejemplo del dataset.

**Nº de muestras:** Las diferencias más significativas cuando variamos el número de muestras se encuentran entre el dataset con 1 y 10 muestras, mejorando del 70.2 % al

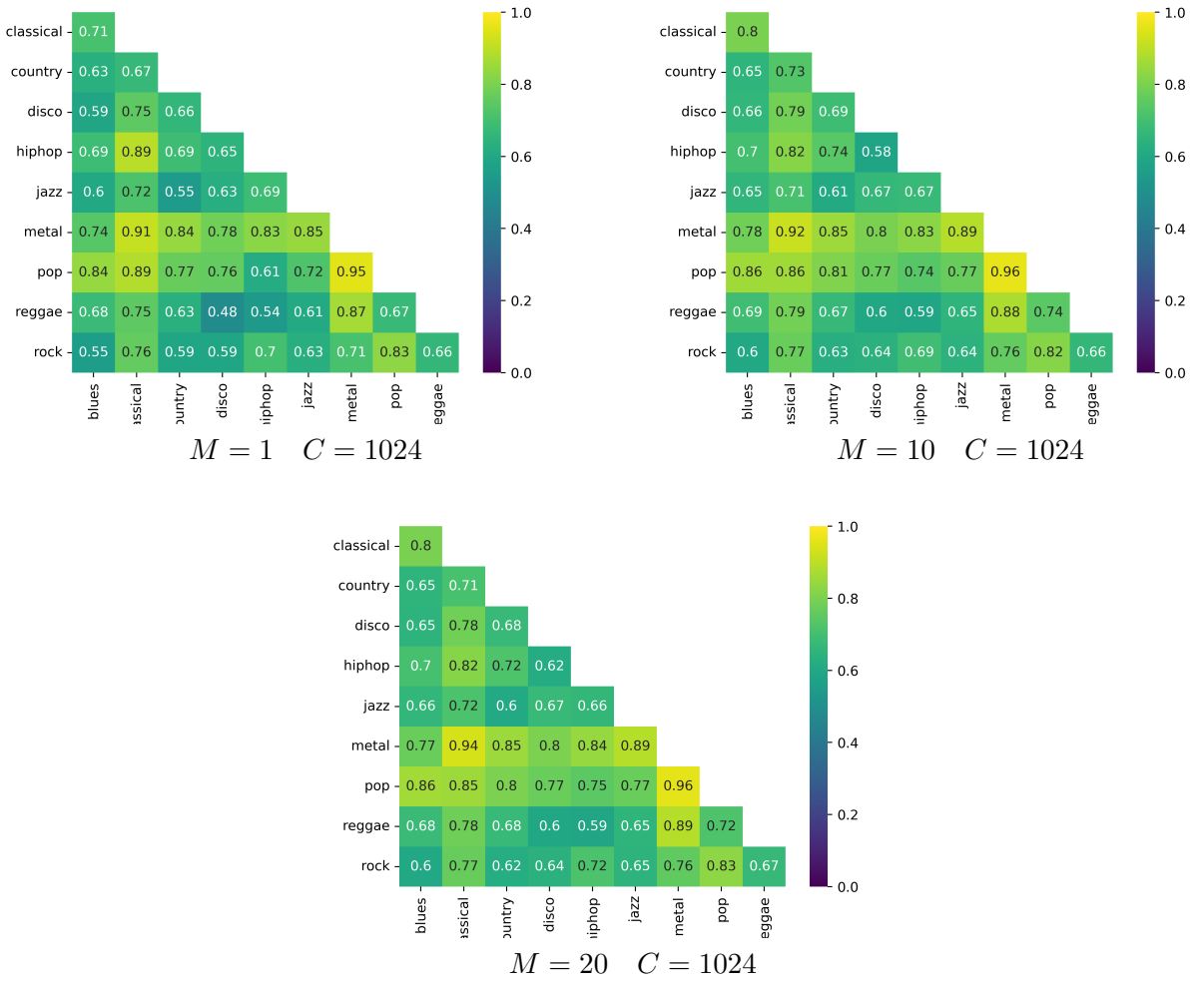


Figura 4.6: Accuracies de los entrenamientos de las QCNNs. $M$ indica el número de muestras y $C$ hace referencia al número de características de cada ejemplo del dataset.

73.6 %, mientras que mantiene resultados similares al duplicar las muestras de 10 a 20. Agrupando por este parámetro todas las ejecuciones realizadas, podemos ver a simple vista estas diferencias, resumidas en la tabla 4.4.

| Muestras | $\mu$ | $\sigma$ |
|---|---|---|
| 1 | 70.2 % | 0.112 |
| 10 | 73.6 % | 0.095 |
| 20 | 73.5 % | 0.094 |
| $\Delta_{1,10}$ | 0.034 | - |
| $\Delta_{10,20}$ | -0.001 | - |

Tabla 4.4: (QCNN) Resumen de resultados por cantidad de muestras por audio.

Esto se puede ver claramente con los géneros disco-jazz, country-disco y country-jazz, encontrando mejoras de hasta 15.5 puntos en porcentaje entre accuracies, como muestra la tabla 4.5:

| | disco vs jazz | | | country vs disco | | | country vs jazz | | |
|---|---|---|---|---|---|---|---|---|---|
| | 256 | 512 | 1024 | 256 | 512 | 1024 | 256 | 512 | 1024 |
| 1 | 61.0 % | 53.0 % | 62.7 % | 56.7 % | 60.7 % | 66.0 % | 48.7 % | 51.7 % | 54.7 % |
| 10 | **68.3 %** | **68.5 %** | 66.8 % | **69.5 %** | **69.0 %** | **68.9 %** | **61.1 %** | **64.4 %** | **60.8 %** |
| 20 | 67.0 % | 67.8 % | **67.3 %** | 68.6 % | 68.7 % | 67.6 % | 60.3 % | 63.0 % | 60.4 % |
| $\Delta_{1,10}$ | 0.073 | **0.155** | 0.041 | **0.129** | 0.084 | 0.029 | 0.125 | **0.128** | 0.062 |
| $\Delta_{10,20}$ | -0.013 | -0.008 | 0.006 | -0.009 | -0.003 | -0.014 | -0.009 | -0.014 | -0.004 |
| $CV_1$ | 0.175 | 0.110 | 0.123 | 0.170 | 0.096 | 0.079 | 0.165 | 0.142 | 0.073 |
| $CV_{10}$ | **0.030** | **0.020** | 0.025 | 0.032 | 0.040 | **0.020** | **0.028** | 0.048 | **0.046** |
| $CV_{20}$ | 0.037 | 0.025 | **0.023** | **0.026** | **0.033** | 0.026 | 0.031 | **0.016** | 0.053 |

Tabla 4.5: (QCNN) Comparación de resultados por número de muestras para las parejas disco-jazz, country-disco y country-jazz. La máxima diferencia entre 1 y 10 muestras se encuentra en disco-jazz con 0.155, calculado sobre todas las parejas de géneros.

En cambio, duplicar el número de muestras resulta perjudicial en estos 3 casos particulares de géneros, donde se empeora la precisión del modelo en la mayoría de casos, exceptuando la ejecución en disco-jazz con 1024 características a partir de los mapas de calor de la figura 4.6.

Por otro lado, también se puede observar el impacto de la forma de obtener datos del conjunto de audios con la desviación típica que muestra la tabla 4.4. Además de presentar mejores resultados en general, los modelos con mayor número de audios también se comportan de manera más robusta a la inicialización aleatoria de parámetros, pasando de una desviación de 0.112 con 1 muestra a desviaciones alrededor de 0.095 tanto para 10 y 20 muestras. Esto también queda confirmado con las 3 últimas filas de la tabla 4.5, donde los coeficientes de variación para 10 y 20 muestras exponen valores mucho más pequeños en comparación con los calculados con 1 muestra, indicando una mejora en la efectividad del modelo, menor variabilidad en los entrenamientos o una combinación de ambos.

**Nº de características:** Cambiando ahora de eje de comparación, en este caso resulta más complicado obtener conclusiones a simple vista entre los distintos números de características.

| Características | $\mu$ | $\sigma$ |
|---|---|---|
| 256 | 0.719 | 0.103 |
| 512 | 0.727 | 0.102 |
| 1024 | 0.726 | 0.101 |
| $\Delta_{256,512}$ | 0.007 | - |
| $\Delta_{256,512}$ | -0.001 | - |

Tabla 4.6: (QCNN) Resumen de resultados por resolución de imagen.

Comparando las tablas 4.4 y 4.6, no se puede concluir a priori una mejoría en el modelo con el aumento de características ya que, por un lado, la media de los accuracies comprende valores muy similares (diferencia máxima de 0.08 en porcentaje),

y por otro, las desviaciones típicas pueden ser explicadas por las variaciones en los resultados con los distintos números de muestras.

Buscando más información en las ejecuciones por pares de géneros, las diferencias más destacables se pueden ver en los datasets con menos muestras (1), debido a la reducida cantidad de información disponible.

| | blues vs jazz | | | jazz vs pop | | | pop vs rock | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 10 | 20 | 1 | 10 | 20 | 1 | 10 | 20 |
| 256 | 56.0 % | 65.4 % | 66.0 % | 69.0 % | 75.3 % | 76.7 % | 77.7 % | 81.8 % | 81.6 % |
| 512 | **64.7 %** | **67.5 %** | **66.7 %** | **76.0 %** | **77.9 %** | **77.4 %** | **84.7 %** | **83.1 %** | **83.2 %** |
| 1024 | 60.0 % | 64.0 % | 65.7 % | 72.0 % | 76.6 % | 77.3 % | 83.0 % | 81.8 % | 82.9 % |
| $\Delta_{256,512}$ | **0.087** | 0.021 | 0.007 | **0.070** | 0.026 | 0.007 | **0.070** | 0.012 | 0.016 |
| $\Delta_{256,512}$ | -0.047 | -0.021 | -0.010 | -0.040 | -0.013 | 0.000 | -0.017 | -0.013 | -0.003 |
| $CV_{256}$ | 0.154 | 0.042 | 0.020 | 0.111 | 0.037 | 0.027 | 0.062 | 0.015 | 0.009 |
| $CV_{512}$ | 0.086 | **0.023** | **0.014** | 0.101 | 0.026 | 0.024 | 0.060 | **0.015** | **0.005** |
| $CV_{1024}$ | **0.083** | 0.043 | 0.029 | **0.097** | **0.008** | **0.019** | 0.076 | 0.020 | 0.010 |

Tabla 4.7: (QCNN) Comparación de resultados por número de características de las parejas blues-jazz, jazz-pop y pop-rock.

Estas diferencias se ven en combinaciones como jazz-pop, pop-rock y blues-jazz aunque no son tan notables como en el caso anterior, viendo variaciones muy amplias entre 1 muestra y las de 10 y 20 en los 3 casos en la tabla 4.7, indicando que la efectividad del circuito no mejora con el aumento de características. Esto se puede explicar desde 2 puntos de vista:

- Se aumenta la probabilidad de encontrarnos con el problema de las *barren plateaus*, haciendo que la función de pérdida sea más difícil de navegar y el entrenamiento se estanque debido al incremento de qubits.
- Este incremento también induce a un mayor número de relaciones entre las distintas características, con lo que también es necesario incrementar la expresividad del modelo, de tal manera que este sea capaz de capturar la esencia de todas esas nuevas conexiones. Un incremento de 17 parámetros (esto es, un 33.3 % más que el circuito usado para 8 qubits) resulta relativamente pequeño en comparación con el uso del doble de características.

Estas observaciones pueden indicar que los modelos para 512 y 1024 características no son lo suficiente expresivos para generalizar de manera adecuada de los datos y, por tanto, sea necesario un cambio de arquitectura o un aumento en el número de parámetros (como se sugiere en [8]).

**Por género:** La tabla 4.8 se ha obtenido calculando la media de las accuracies en función del género, mostrando el resultado en porcentaje. En esta tabla se puede ver una clara tendencia en la efectividad del modelo con una arquitectura central de 9 qubits, haciendo uso de 512 características y 10 muestras, consiguiendo el mejor resultado en 6 de los 10 géneros del dataset y con precisiones cercanas a las mejores obtenidas en cada caso.

| C | M | blues | classical | country | disco | hiphop | jazz | metal | pop | reggae | rock |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 256 | 1 | 66.9 % | 77.4 % | 64.0 % | 63.5 % | 70.4 % | 64.4 % | 82.3 % | 78.5 % | 65.5 % | 64.6 % |
| | 10 | 70.5 % | 78.9 % | 70.3 % | 69.0 % | 70.3 % | 68.8 % | 84.7 % | 80.4 % | 68.4 % | 68.7 % |
| | 20 | 70.4 % | 78.7 % | 69.7 % | 68.4 % | 70.5 % | 69.1 % | 85.0 % | 80.5 % | 69.1 % | 69.1 % |
| 512 | 1 | 67.2 % | 77.6 % | 64.4 % | 64.6 % | 68.9 % | 64.8 % | 82.7 % | 78.8 % | 65.2 % | 66.0 % |
| | 10 | **72.0 %** | **79.9 %** | **71.3 %** | 69.6 % | **71.5 %** | 69.9 % | 85.1 % | **82.2 %** | **70.2 %** | 69.4 % |
| | 20 | 71.5 % | 79.4 % | 70.7 % | **69.5 %** | 71.2 % | **70.0 %** | 85.4 % | 81.7 % | 69.9 % | **70.0 %** |
| 1024 | 1 | 67.0 % | 78.3 % | 67.0 % | 65.4 % | 69.8 % | 66.4 % | 83.1 % | 78.3 % | 65.4 % | 67.0 % |
| | 10 | 71.1 % | 79.8 % | 70.8 % | 68.8 % | 70.7 % | 69.6 % | 85.1 % | 81.3 % | 69.7 % | 69.0 % |
| | 20 | 70.7 % | 79.6 % | 70.0 % | 68.9 % | 71.3 % | 69.7 % | **85.5 %** | 81.3 % | 69.4 % | 69.3 % |

Tabla 4.8: Medias del accuracy en el conjunto de test por género en QCNN. $C$ es el número de características y $M$ el número de muestras por audio.

En la línea de las observaciones anteriores sobre el aumento en el número de muestras, aquí también se puede apreciar esta mejoría por género, ya que prácticamente en la totalidad de los casos la precisión del modelo se ve incrementada si comparamos los resultados de 1 y 10 muestras. Los únicos géneros que no presentan mejoría significativa se corresponde con la música clásica y el hiphop.

En cuanto al número de características, se observa una menor variación si comparamos los valores obtenidos sobre todos los modelos propuestos, con una diferencia máxima en 3 puntos de porcentaje (como, por ejemplo, en country con 512 y 1024 características).

Para terminar con el análisis por géneros, se puede ver una tendencia clara en los accuracies de ciertos géneros musicales. Los 3 conjuntos más distinguibles del resto se corresponden con la música clásica, el metal y el pop, donde se obtienen accuracies superiores al 75 % en todos los tipos de dataset.

Una vez se han estudiado los resultados atendiendo a estas de características, se pasa a un análisis más general, intentando obtener conclusiones sobre la esencia del dataset y los modelos. La tabla 4.9 muestra las medias de los accuracies entre todas las parejas del dataset respecto al número de muestras y características, junto con sus coeficientes de variación en cada caso.

| $\mu \pm \sigma$ | 1 | 10 | 20 |
|---|---|---|---|
| 256 | 69.8 % ± 6.4 | 73.0 % ± 1.9 | 73.1 % ± 1.5 |
| 512 | 70.0 % ± 7.0 | **74.1 % ± 1.7** | 73.9 % ± 1.4 |
| 1024 | 70.8 % ± 5.6 | 73.6 % ± 1.9 | 73.6 % ± 1.4 |

| CV | 1 | 10 | 20 |
|---|---|---|---|
| 256 | 0.092 | 0.026 | 0.020 |
| 512 | 0.099 | 0.023 | **0.019** |
| 1024 | 0.079 | 0.026 | 0.019 |

Tabla 4.9: (QCNN) Resumen de métricas calculadas para cada tipo de experimento con todos los pares de géneros. A la izquierda, media y desviación típica. A la derecha, coeficiente de variación.

Estas tablas resumen las conclusiones descritas anteriormente: disminución de la desviación típica y aumento de la efectividad con el aumento de muestras, pero poca mejoría con el incremento de resolución de las imágenes de input.

Por último, al igual que el sobreajuste es un problema que aparece comúnmente en modelos de ML, es preciso analizar si este fenómeno también está presente en los circuitos y modelos cuánticos. Para ello se obtienen las gráficas del accuracy por medio de la herramienta `wandb`, que automáticamente *loggea* en cada iteración estas métricas para su posterior estudio. Al igual que en la elección del número de épocas (sección 4.1), se eligen las parejas country-jazz, blues-classical y metal-pop para dar una muestra de todo el rango de parejas posibles. La figura 4.7 muestra esta tendencia para estas tres parejas de géneros musicales utilizando el dataset con 1 muestra y 256 características.

Empezando por la primera de ellas, country-jazz, se puede observar como hay una clara diferencia entre los accuracies en el conjunto de entrenamiento y en el de test (15 puntos en porcentaje), sugiriendo que puede presentar sobreajuste a los datos de entrenamiento.

En la segunda gráfica, blues-classical, la separación entre el accuracy de train y test no es tan pronunciada como en el caso anterior (7.1 puntos en porcentaje), pero sí que presentan tendencias similares a lo largo del entrenamiento. Aun así, en otros pares de géneros con accuracies similares, estas diferencias pueden incrementarse como en disco-metal o reducirse como blues-pop como se puede ver en la figura 4.8.

En la última de las comparaciones de la figura 4.7 se puede ver como el modelo es muy efectivo a la hora de aprender de los datos de entrenamiento, en donde las curvas de test y train se comportan prácticamente de la misma manera. Esto muestra también que este tipo de modelos puede aprender de forma muy efectiva de una cantidad limitada de datos.

### 4.2.2. CNNs

Al igual que los resultados principales mostrados en 4.2.1, se presentan los mapas de calor de los experimentos con CNNs en la figura 4.10.

A simple vista, lo primero que se puede destacar es la mejora paulatina con el uso de mayor cantidad de muestras, acercándose a accuracies del 75 % con 10 muestras y superiores al 80 % en el caso de 20 fragmentos por audio. Estos comentarios preliminares quedarán mejor representados con un análisis más riguroso en el resto de esta sección:

**Nº de muestras:** De la misma manera que se ha observado para el caso de las arquitecturas cuánticas, el aumento de datos de entrenamiento también mejora la efectividad de la CNN propuesta, como se ve en la tabla 4.10 con un aumento máximo de 9.3 puntos entre el dataset con 1 y 10 muestras.

| Muestras | $\mu$ | $\sigma$ |
|---|---|---|
| 1 | 71.4 % | 0.105 |
| 10 | 79.1 % | 0.088 |
| 20 | 80.7 % | 0.085 |
| $\Delta_{1,10}$ | 0.078 | 0.044 |
| $\Delta_{10,20}$ | 0.016 | 0.026 |

Tabla 4.10: (CNN) Resumen de resultados por cantidad de muestras por audio.

En este caso, a diferencia de las QCNNs, sí que se aprecia una ligera diferencia entre el accuracy medio para los datos con 10 y 20 muestras. Esto puede indicar que el modelo clásico propuesto es un poco más expresivo y tiene mayor potencial que su contrapartida cuántica.

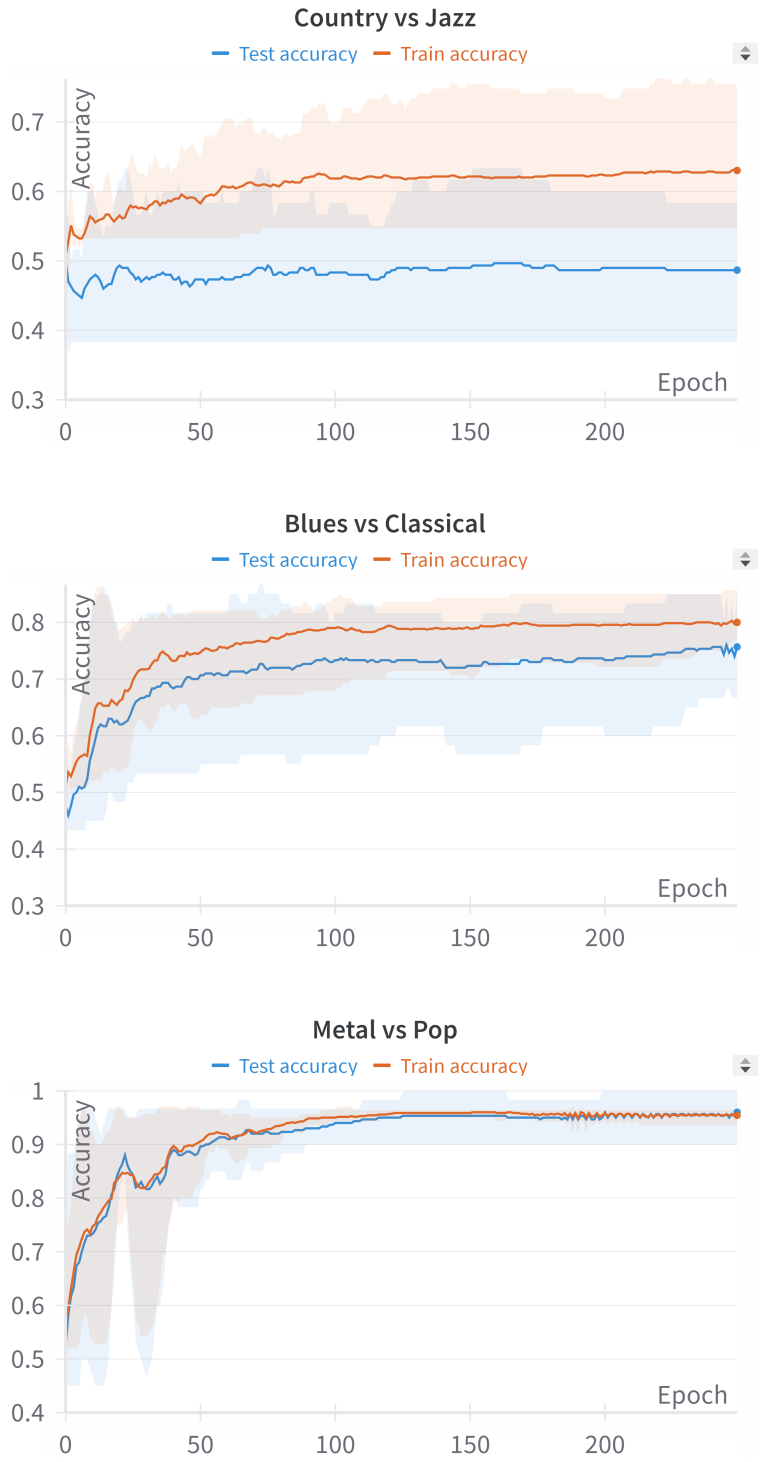


Figura 4.7: (QCNN) Tendencia de los accuracies a lo largo del entrenamiento para diferentes los géneros country-jazz, blues-classical y metal-pop (1).

Además, se ha observado que en todos los géneros, este aumento en los datos (de 1 a 10 muestras) ha provocado un aumento en el accuracy final. Esto incluye la pareja de géneros metal-pop con un ligero ascenso de 96.1 % a 97.0 %. La tabla 4.11 resume

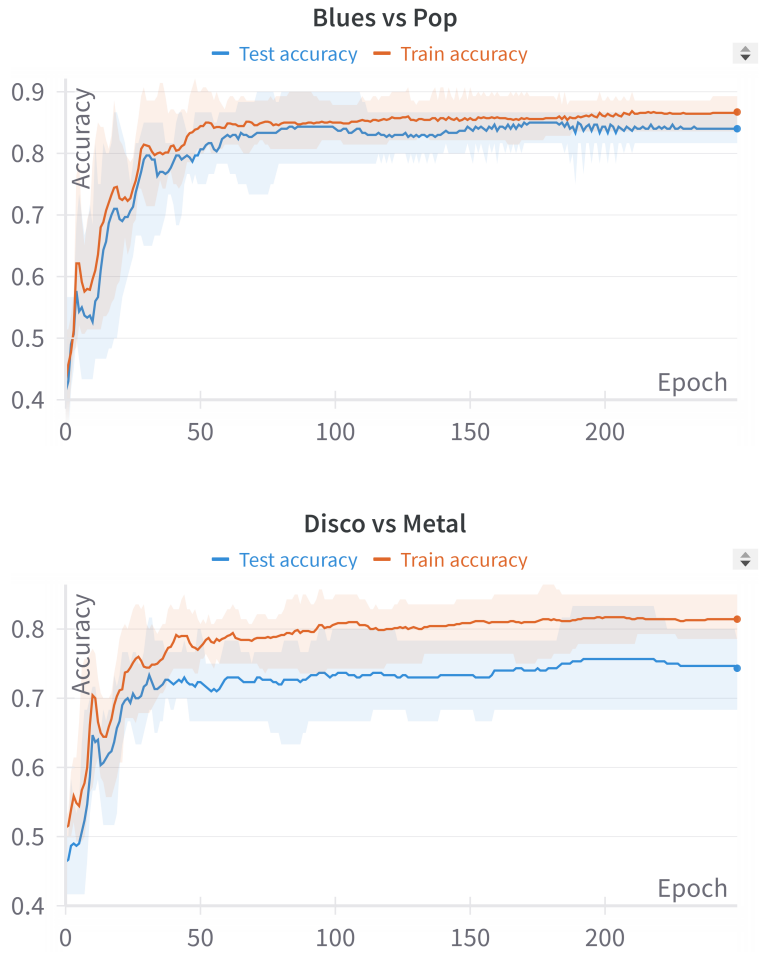


Figura 4.8: (QCNN) Tendencia de los accuracies a lo largo del entrenamiento para los géneros blues-pop y disco-metal (2).

algunos resultados de las parejas blues-rock, jazz-pop y disco-pop, tratándose de los géneros que más se han beneficiado de este tipo de variación, alcanzando hasta 19.5 puntos de subida en jazz-pop.

**Nº de características:** En cuanto a la resolución de las imágenes, se puede ver en la tabla 4.12 un comportamiento similar al observado en el caso de las QCNNs (tabla 4.6), donde apenas se aprecia una mejoría en el modelo al incrementar el número de características de los datos.

La tabla 4.13 muestra los resultados para los pares de géneros blues-classical, blues-country y blues-disco, que más han sido favorecidos por este aumento en las características.

El mayor cambio se ha visto en la pareja blues-classical con una diferencia al alza de 11.9 en el caso de las 10 muestras entre las 256 y 512 características. A su vez, esta pareja también muestra un decremento de 7 puntos para el caso de 1 muestra, pudiendo indicar la necesidad de realización de más ejecuciones para verificar estos resultados.

Por otro lado, centrando nuestra atención en el coeficiente de variación, en la mayoría

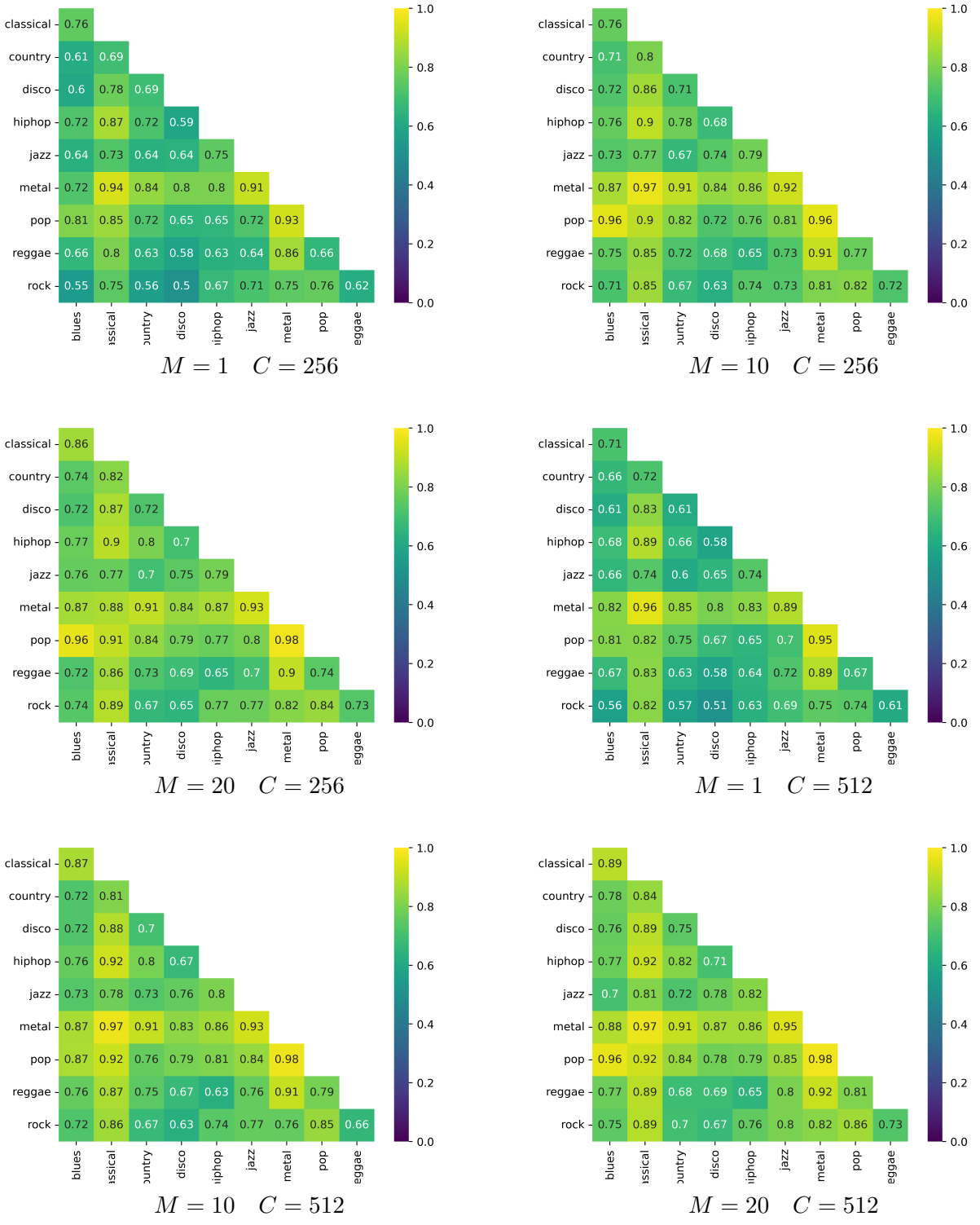


Figura 4.9: Accuracies de los entrenamientos de las CNNs. $M$ indica el número de muestras y $C$ hace referencia al número de características de cada ejemplo del dataset.

de casos se observan los valores más reducidos en el caso de las 512 características, mientras que las de menor resolución (256) presenten picos más elevados como 0.174

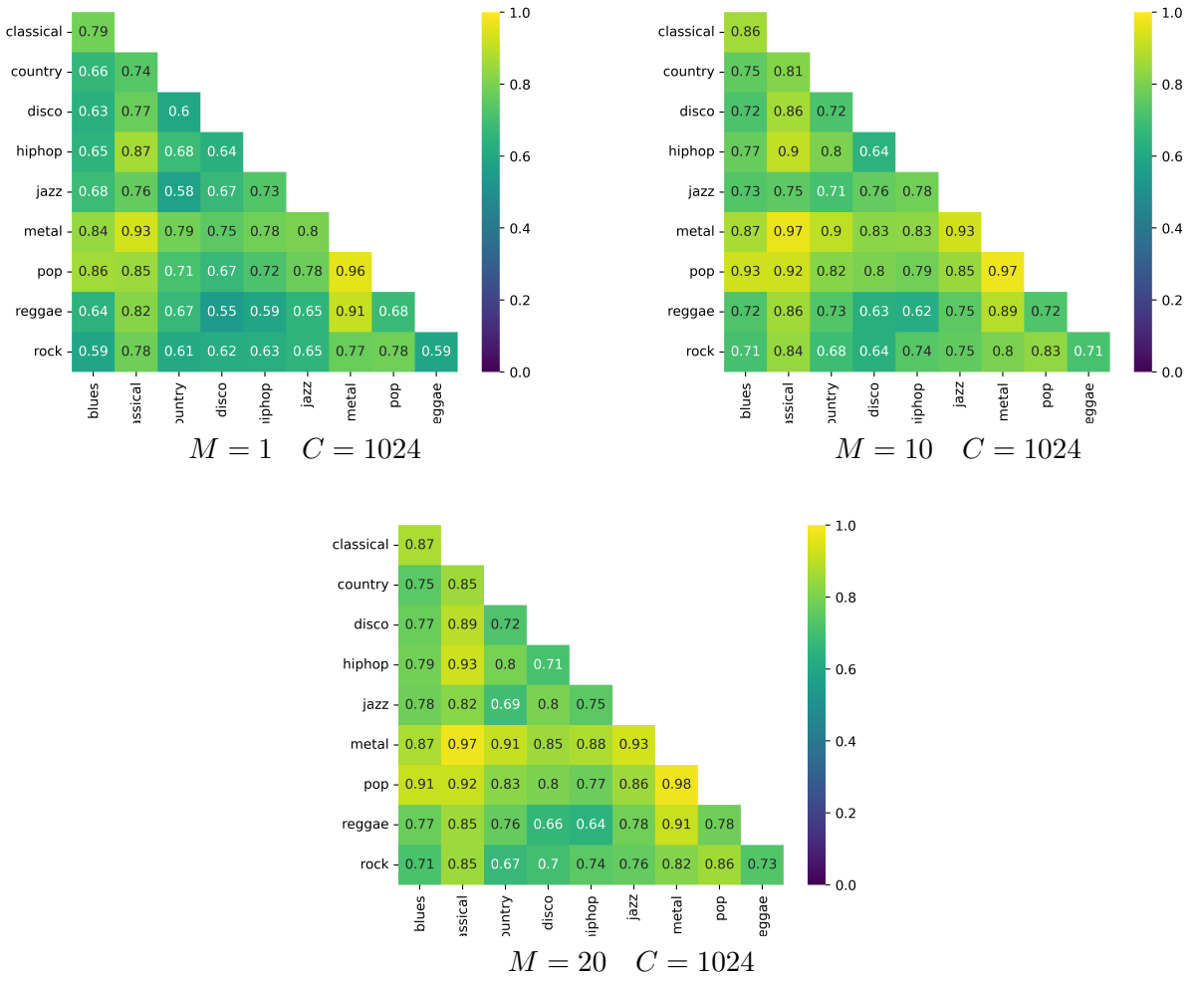


Figura 4.10: Accuracies de los entrenamientos de las CNNs. $M$ indica el número de muestras y $C$ hace referencia al número de características de cada ejemplo del dataset.

| | blues vs rock | | | jazz vs pop | | | disco vs pop | | |
|---|---|---|---|---|---|---|---|---|---|
| | 256 | 512 | 1024 | 256 | 512 | 1024 | 256 | 512 | 1024 |
| 1 | 55.3 % | 56.0 % | 58.7 % | 61.3 % | 70.0 % | 77.7 % | 61.0 % | 67.3 % | 66.3 % |
| 10 | 72.5 % | 71.7 % | 70.5 % | **81.1 %** | 84.5 % | 84.2 % | 72.7 % | **79.7 %** | 80.1 % |
| 20 | **73.5 %** | **74.8 %** | **71.4 %** | 80.0 % | **85.3 %** | **85.7 %** | **78.5 %** | 78.3 % | **80.5 %** |
| $\Delta_{1,10}$ | **0.172** | 0.157 | 0.118 | **0.198** | 0.145 | 0.065 | 0.117 | 0.123 | **0.138** |
| $\Delta_{10,20}$ | 0.010 | 0.031 | 0.009 | -0.011 | 0.008 | 0.015 | 0.059 | -0.014 | 0.004 |
| $CV_1$ | 0.145 | 0.132 | 0.101 | 0.195 | 0.108 | 0.056 | 0.171 | 0.122 | 0.136 |
| $CV_{10}$ | **0.034** | 0.048 | 0.062 | **0.047** | 0.046 | 0.022 | 0.085 | **0.027** | 0.034 |
| $CV_{20}$ | 0.052 | **0.010** | **0.060** | 0.078 | **0.027** | **0.014** | **0.020** | 0.076 | **0.013** |

Tabla 4.11: (CNN) Comparación de resultados por número de muestras por audio de blues-rock, jazz-pop y disco-pop.

| Características | $\mu$ | $\sigma$ |
|---|---|---|
| 256 | 76.3 % | 0.102 |
| 512 | 77.6 % | 0.103 |
| 1024 | 77.3 % | 0.100 |
| $\Delta_{256,512}$ | 0.013 | - |
| $\Delta_{512,1024}$ | -0.003 | - |

Tabla 4.12: (CNN) Resumen de resultados por características por dato.

| | blues vs classical | | | blues vs country | | | blues vs disco | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 10 | 20 | 1 | 10 | 20 | 1 | 10 | 20 |
| 256 | 78.0 % | 76.1 % | 85.6 % | 59.0 % | 70.2 % | 73.8 % | 62.7 % | 72.4 % | 71.9 % |
| 512 | 71.0 % | **88.0 %** | **88.8 %** | 66.3 % | 72.0 % | **77.5 %** | 61.0 % | **73.6 %** | 75.4 % |
| 1024 | **78.7 %** | 86.8 % | 87.4 % | **67.0 %** | **74.9 %** | 74.3 % | **63.3 %** | 71.6 % | **76.8 %** |
| $\Delta_{256,512}$ | -0.070 | 0.119 | 0.032 | 0.073 | 0.019 | 0.037 | -0.017 | 0.012 | 0.035 |
| $\Delta_{512,1024}$ | 0.077 | -0.012 | -0.014 | 0.007 | 0.028 | -0.032 | 0.023 | -0.019 | 0.014 |
| $CV_{256}$ | 0.067 | 0.174 | 0.041 | 0.107 | **0.048** | 0.054 | 0.144 | 0.074 | 0.179 |
| $CV_{512}$ | 0.086 | 0.026 | **0.018** | **0.048** | 0.049 | **0.017** | **0.096** | **0.041** | 0.045 |
| $CV_{1024}$ | **0.059** | **0.023** | 0.023 | 0.062 | 0.052 | 0.032 | 0.110 | 0.089 | **0.018** |

Tabla 4.13: (CNN) Comparación de resultados por número de características de blues-classical, blues-country y blues-disco.

en blues-classical o 0.144 y 0.179 en blues-disco. Este comportamiento sugiere que, mientras que no se puede apreciar una mejora general con el aumento de características (como muestra la tabla 4.12), la variación de los resultados es menor y, por tanto, un entrenamiento más estable y una menor influencia de la inicialización aleatoria de los parámetros.

**Por géneros:** En el caso de las CNNs, también se puede ver en la tabla 4.14 una clara superioridad de la eficacia de la CNN con el uso de 512 características y 20 muestras, predominando sobre el resto de alternativas y obteniendo los mejores resultados en la gran mayoría de casos.

De manera general, se observan patrones de aprendizaje similares a los vistos en las QCNNs con respecto a los géneros musicales, donde el metal, classical y pop son los tipos de música más identificables y distinguibles.

En cambio, para este tipo de modelo sí que se aprecia un incremento en el accuracy en todo el rango de posibilidades incluido para los casos de classical y hiphop, como no ocurría en la variante cuántica.

Pasando ahora a una visión más global de todos los experimentos con CNNs, la tabla 4.15 recopila las métricas de precisión y desviación típica por tipo de experimento, además de sus coeficientes de variación.

En esta tabla se puede volver a poner de manifiesto las ideas comentadas anteriormente con respecto al número de características en la bondad del modelo. La máxima diferencia en este aspecto se ve con 256 y 512 features con el tamaño del dataset más grande (20

| C | M | blues | classical | country | disco | hiphop | jazz | metal | pop | reggae | rock |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 256 | 1 | 67.5 % | 79.7 % | 66.6 % | 64.1 % | 69.7 % | 66.7 % | 82.4 % | 75.4 % | 67.0 % | 65.5 % |
| | 10 | 77.6 % | 85.0 % | 75.5 % | 73.1 % | 76.8 % | 76.6 % | 89.3 % | 83.4 % | 75.4 % | 74.5 % |
| | 20 | 79.3 % | 86.1 % | 76.8 % | 74.7 % | 78.1 % | 77.5 % | 88.9 % | 84.8 % | 74.9 % | 76.6 % |
| 512 | 1 | 68.3 % | 80.9 % | 67.0 % | 64.9 % | 70.1 % | 70.8 % | 85.3 % | 75.3 % | 69.3 % | 65.4 % |
| | 10 | 78.4 % | 87.7 % | 76.2 % | 74.2 % | 77.6 % | 79.0 % | 89.0 % | 84.7 % | 75.4 % | 73.8 % |
| | 20 | **80.8 %** | **89.0 %** | **78.2 %** | 76.5 % | **78.8 %** | **80.1 %** | 90.5 % | **86.6 %** | **77.3 %** | **77.5 %** |
| 1024 | 1 | 70.6 % | 81.0 % | 67.0 % | 65.3 % | 69.7 % | 69.6 % | 83.8 % | 77.9 % | 67.7 % | 66.8 % |
| | 10 | 78.1 % | 86.7 % | 76.9 % | 73.4 % | 76.6 % | 77.7 % | 88.5 % | 84.8 % | 73.5 % | 74.5 % |
| | 20 | 80.4 % | 88.4 % | 77.6 % | **76.5 %** | 77.6 % | 79.9 % | **90.5 %** | 85.6 % | 76.6 % | 76.0 % |

Tabla 4.14: Medias de accuracies en test por género en CNNs.

| $\mu \pm \sigma$ | 1 | 10 | 20 |
|---|---|---|---|
| 256 | 70.5 % ± 7.2 | 78.7 % ± 2.9 | 79.8 % ± 3.4 |
| 512 | 71.7 % ± 6.1 | 79.6 % ± 3.4 | **81.5 % ± 3.5** |
| 1024 | 71.9 % ± 7.1 | 79.1 % ± 3.4 | 80.9 % ± 4.0 |

| CV | 1 | 10 | 20 |
|---|---|---|---|
| 256 | 0.104 | 0.037 | 0.044 |
| 512 | 0.088 | 0.045 | **0.028** |
| 1024 | 0.101 | 0.045 | 0.032 |

Tabla 4.15: (CNN) Resumen de métricas para cada tipo de experimento calculada con todos los pares de géneros. A la izquierda, media y desviación típica. A la derecha, coeficiente de variación.

muestras) con un incremento de 1.7 puntos, con lo que se encuentra dentro de la dispersión respecto de la media entre ambos casos ($\sigma \geq 3{,}4$).

Poniendo el foco en los coeficientes de variación, se aprecia una ligera mejoría en el caso de la extracción de 1 y 20 muestras para 256 y 512 características, verificando en cierta manera las ideas planteadas en el análisis por características anterior. En cualquiera de los casos para 1024 características, los resultados siempre tienden a alcanzar valores más elevados que los vistos con la mitad de ellas (512).

Como último detalle a tener en cuenta con respecto a las CNNs, es importante comprobar si muestra signos de overfitting a lo largo del entrenamiento. Para ello se vuelven a elegir los géneros country-jazz, blues-classical y metal-pop con los experimentos de 256 características y 1 muestra como en el caso cuántico. La figura 4.11 ilustra el progreso del accuracy del modelo clásico con estos 3 pares de géneros.

Es claro que las dos primeras gráficas, country-jazz y blues-classical, muestran grandes diferencias entre los accuracies de entrenamiento y test. En estos casos, llegados al final de la ejecución, las partes sombreadas que comprenden la desviación típica se encuentran totalmente disjuntas indicando un fuerte efecto de sobreajuste en los datos. En particular, en la mayoría de estos casos el accuracy en el conjunto de entrenamiento alcanza el valor máximo de 1.

El único caso donde estas tendencias son más equiparables entre estas métricas las encontramos en el par metal-pop, aunque también se observe un aplanamiento en la curva alrededor de la época 120 para el conjunto de test.

Por último, en la figura 4.12 aparecen el progreso en el entrenamiento para blues-pop y disco-metal, mostrando tendencias similares.

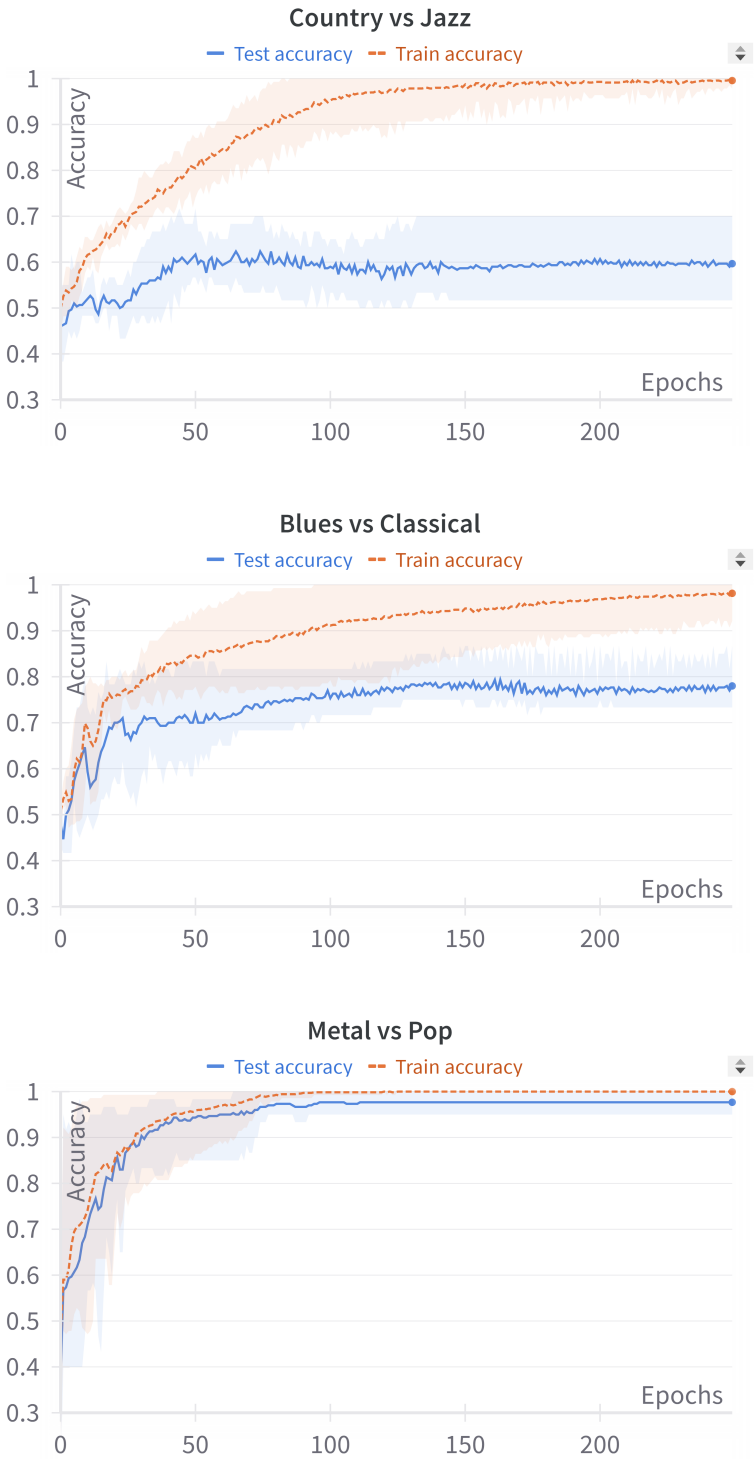


Figura 4.11: (CNN) Tendencia de los accuracies a lo largo del entrenamiento para los géneros country-jazz, blues-classical y metal-pop (1).

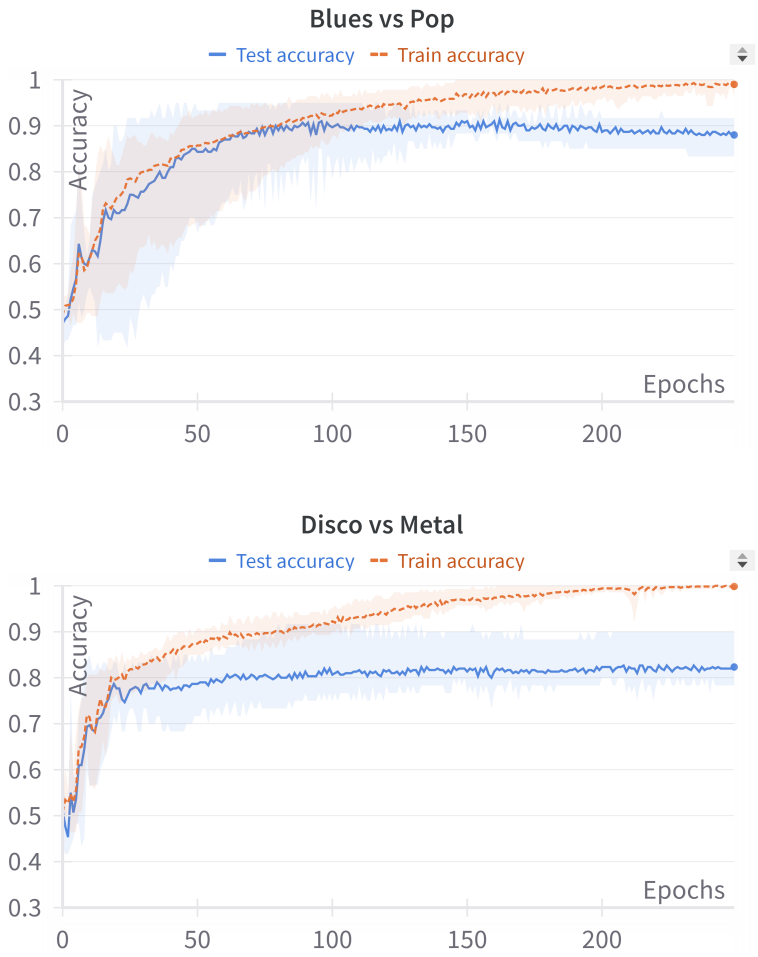


Figura 4.12: (CNN) Tendencia de los accuracies a lo largo del entrenamiento para los géneros blues-pop y disco-metal (2).

Capítulo 5

# Conclusiones y Trabajo Futuro

## 5.1. Conclusiones

Como se ha observado en los resultados mostrados, las QCNNs aquí propuestas y probadas no están todavía a la altura de redes convolucionales relativamente sencillas, como muestra la figura 5.1, indicando el largo camino que queda por recorrer en el campo del QML.

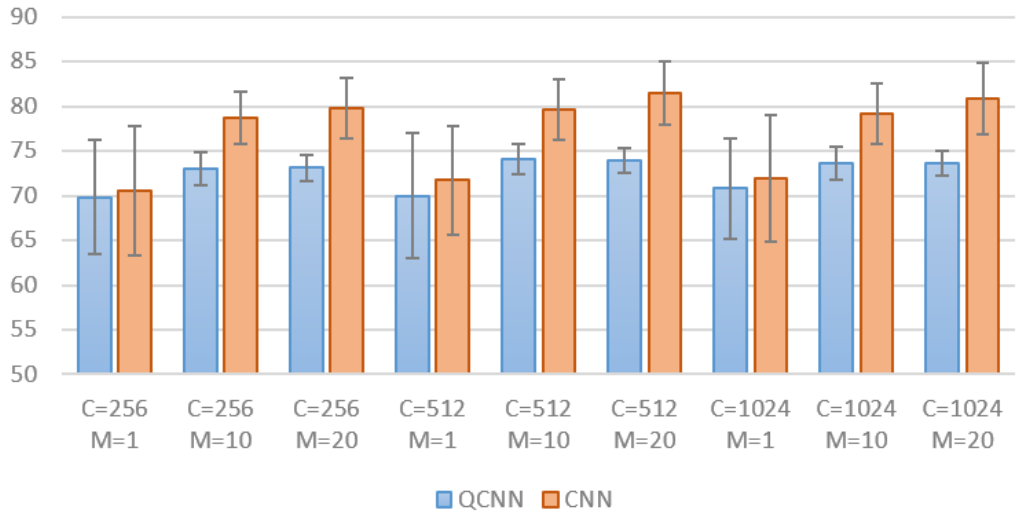


Figura 5.1: Gráfica de comparación entre los accuracies globales de QCNNs y CNN para cada tipo de preprocesado: $C$ para el número de características y $M$ para el número de muestras por audio. Los datos provienen de las tablas 4.9 y 4.15.

Sin embargo, a pesar de que estos resultados sean a priori desalentadores, es preciso tener en cuenta una serie de cuestiones:

**Número de parámetros:** Lo primero importante a destacar es el tamaño y complejidad de cada tipo de modelo en la ejecución de cada experimento. Aunque las QCNNs y las CNN no puedan compararse de manera canónica, el número de parámetros entrenables es una característica que ambos modelos comparten. La tabla 5.1 resume la cantidad de estos parámetros que presenta cada circuito y modelo probado en este trabajo.

| | QCNN | | | CNN | | |
|---|---|---|---|---|---|---|
| Características | 256 | 512 | 1024 | 256 | 512 | 1024 |
| **Nº Parámetros** | 51 | **68** | 68 | 8772 | **29572** | 61572 |

Tabla 5.1: Parámetros entrenables por tipo de experimento realizado.

Si tenemos en cuenta las tablas 4.9 y 4.15, podemos ver como de manera global, el mejor resultado de los modelos cuánticos propuestos obtiene un 74.1 % de efectividad, mientras que la CNN con mejor rendimiento obtiene un 81.5 %.

Esta diferencia de 7.4 puntos entre estos modelos es considerable a simple vista, pero el incremento en el número de parámetros por parte del modelo clásico es también significativamente mayor: un 33 % en el caso cuántico, pero más del triple para la CNN.

Por otra parte, para el dataset con menor cantidad de datos de entrenamiento (1 muestra y 256 características), los resultados entre cada paradigma son muy similares, obteniendo un 70.5 % de accuracy para las CNN y 69.8 % para la QCNN. De esta manera, se ha cumplido uno de los objetivos comentados en el capítulo 3, en el que se demuestra una menor necesidad en el número de parámetros entrenables en los circuitos cuánticos para obtener resultados comparables que las CNNs, en la que el modelo clásico utiliza 172 veces más parámetros que el modelo cuántico.

**Diferencias en el learning rate:** Un detalle que resulta curioso reside en la pequeña diferencia en la elección del *learning rate* en cada tipo de modelo, necesitando los modelos cuánticos un valor más pequeño ($lr = 0{,}01$) que el clásico ($lr = 0{,}001$).

Esta pequeña diferencia se puede explicar recurriendo a una cualidad física que aparece solo en los modelos cuánticos: el entrelazado. Tal y como sugieren en [10], las puertas lógicas cuánticas tienen una mayor influencia en el estado global, a pesar de que no actúen en todos los qubits disponibles. De esta manera, no es necesario tener un coeficiente de aprendizaje tan elevado, debido a que una pequeña variación en el parámetro de una puerta del circuito, produce más cambios en el estado cuántico que los que podrían parecer a simple vista.

**Variabilidad de los resultados y coeficiente de variación:** De nuevo, observando las tablas 4.9 y 4.15 vemos como, aunque forma general las CNN presenten un mejor rendimiento que los modelos cuánticos propuestos aquí, los coeficientes de variación obtenidos son superiores en la gran mayoría de los casos. La única excepción se encuentra con el experimento con 512 características y 1 muestra por audio, donde el modelo cuántico produce un $CV = 0{,}099$ mientras que la CNN obtiene un $CV = 0{,}088$.

Esto sugiere que los modelos cuánticos son más estables y tienen entrenamientos con menos variaciones y altibajos, produciendo resultados más robustos que la alternativa clásica.

**Sobreajuste:** Por último, este fenómeno es también importante tenerlo en cuenta para verificar la viabilidad de los modelos propuestos. En este caso, las variantes cuánticas muestran mejores tendencias en las curvas de aprendizaje y comportamientos más estables entre accuracies en los conjuntos de entrenamiento y test.

Esto puede determinar en cierta medida cómo se realizarán los entrenamientos de estos modelos cuánticos en un futuro, con una menor atención en técnicas que traten de minimizar este fenómeno.

Como conclusión final, se puede ver claramente cómo los modelos de QML todavía tienen mucho que aprender, ya que las CNNs consiguen mejores resultados con el aumento en el número de datos.

A pesar de esto, el crecimiento exponencial en la cantidad de parámetros entrenables de las redes clásicas puede favorecer el uso de modelos cuánticos, en los que el número de parámetros crece de manera lineal. Esto da a relucir la efectividad de estos modelos de QML, especialmente con los datasets con un número limitado de datos o características.

## 5.2. Trabajo futuro

Como ya se ha comentado anteriormente, el campo del QML se encuentra en fases tempranas en su desarrollo con lo que existen infinidad de ideas y líneas de trabajo futuro todavía sin explorar. Estas nuevas ideas pueden llevar desde mejoras en los tiempos de entrenamiento, a preprocesados adecuados para representar de mejor manera los datos en los estados cuánticos, al testeo y construcción de nuevos modelos, arquitecturas y estrategias de entrenamiento.

En primer lugar, se puede investigar la realización de offloading a GPU u otros dispositivos con el objetivo de mejorar rendimientos y minimizar tiempos de espera. Además, también se puede estudiar la utilización de multithreading para el testeo de múltiples modelos al mismo tiempo o el uso de librerías de benchmarking, como por ejemplo lazyqml[1]. Por otro lado, el uso de plataformas como `qibo` para la ejecución de circuitos en hardware cuántico real puede dar luz sobre la eficacia actual de estos modelos en un entorno que no es completamente ideal.

Con relación al tratamiento de la información, también se puede seguir indagando en la conexión entre los datos, su preprocesado y los modelos, planteando diversas alternativas:

- Por un lado, el estudio de estos modelos con más datasets relacionados con el audio, como por ejemplo, Yaseen o ICBHI, puede dar pie a avances en el diagnóstico de enfermedades cardiacas y pulmonares.
- Los espectrogramas de Mel son una de las formas más simples para la obtención de una representación visual de una señal sonora, pero no se trata de la única disponible. Los MFCCs (Mel Frequency Cepstral Coefficients) son otro de los tratamientos de la señal más usados y el uso de la DCT se ha observado tener una buena capacidad de compresión de la información, con lo que estudiar el efecto de estas transformaciones en los datos puede resultar interesante si nos centramos en el uso de datos con pocas características o conjuntos de datos de poco tamaño.
- La elección de los 3 segundos de muestra (o ventana) es una elección relativamente arbitraria para la división o extracción de fragmentos de audio. Estudiar otras longitudes de muestras puede dar algo más de información para un buen entrenamiento posterior.

Todas estas variables pueden iluminar el camino hacia un mejor entendimiento sobre la codificación de los datos en estados cuánticos y la relación entre la efectividad de los modelos y la cantidad y calidad de la información de entrenamiento disponible.

[1] https://pypi.org/project/lazyqml/

Centrándonos ahora en las arquitecturas y circuitos cuánticos, se proponen las siguientes ideas. Si se quiere construir circuitos cuánticos que representen de manera más fiel los modelos con CNNs, se puede extender el circuito con más qubits de la siguiente manera (por simplicidad se muestra con solo 2 filtros):

- Duplicar el número de qubits y construir 2 veces el circuito inicial. Posteriormente, con los qubits que deberían devolver el output, se puede utilizar otra o más capas extra que hagan la función de las capas lineales o fully-connected o volver utilizar una estructura en árbol binario hasta que solo quede un único qubit disponible para medir el resultado. Sin embargo, esta idea no es eficiente. Al duplicar el número de qubits, la memoria necesaria para simular este circuito crece exponencialmente.

- Por otro lado, se puede aprovechar la potencia que brinda la computación cuántica añadiendo solo 1 qubit, haciendo que el número de características que se pueden codificar en un estado cuántico se duplique. Con este enfoque, se pueden introducir 2 copias de la imagen como características, haciendo que el circuito aprenda de diferente manera cada copia de las características de la imagen.

  Otra alternativa consiste en la utilización 2 o más capas de convolución contiguas, aumentando de esta manera el número de parámetros y expresividad del modelo sin aumentar el número de qubits.

Para terminar, también se puede proponer el estudio y diseño de modelos híbridos, como por ejemplo, realizar las convoluciones clásicamente pero la clasificación cuánticamente, o viceversa, convoluciones clásicas pero clasificación cuántica.

Chapter 6

# Introduction

In recent years, the development of society and technology has elevated the importance of artificial intelligence (AI), slowly integrating into our daily lives. From text correction and idea suggestions to identifying defects in moving parts, as well as music and movie recommendations, machine learning models are becoming more prevalent in various aspects of life, both at work and during leisure activities on social networks. The rapid pace of technological advancement, along with the enormous amount of information generated by humanity over its evolution, is making it increasingly difficult to manage all this data. Consequently, machine learning models need to keep improving, increasing their complexity and the number of parameters in order to be expressive enough to learn and generalize patterns from this vast amount of information. This, in turn, requires powerful computers with significant computational capacity to train these AI models. This presents a challenge for the continuous development of classical computing, making it necessary to explore other, more efficient models or computational paradigms, such as analog computing, photonic computing, or quantum computing (QC).

The latter, which is the central focus of this work, offers valuable advantages arising from the very nature of quantum physics. Quantum phenomena like entanglement, interference, and superposition allow for faster and more scalable information processing than traditional computing, which is beginning to encounter physical limitations in hardware, such as the end of Moore's law and the physical limits in transistor sizes. As a result, the primary motivation for investigating QC is to provide alternatives to current classical models and explore new ideas in machine learning (ML) inspired by this new paradigm, as it promises significant theoretical improvements in execution times and algorithmic complexity. This leads us to attempt to harness this potential in different problems and tasks, since the amount of information that can be encoded in a set of qubits is much greater than what can be carried by a set of classical bits.

Moreover, we can benefit from the parallelization properties of quantum circuits, allowing us to handle more data simultaneously due to quantum entanglement, enabling work with a large number of features in the data in proportion to the available resources. Additionally, the expressiveness of quantum states could provide better learning with a reduced amount of information. The development of QC as a computational paradigm is being driven by two fronts: quantum hardware and the design of quantum algorithms or circuits. Despite rapid advancements, the hardware component faces many challenges, such as the impact of noise, quantum decoherence, and the difficulty of maintaining quantum states for extended periods. Therefore, hardware development is progressing more slowly than quantum algorithm development.

Continuing in this direction, in the search for alternatives to traditional models that facilitate learning from the vast complexity and volume of information, a potential contender for executing these tasks arises: quantum machine learning (QML). Although current quantum computers and circuits still present significant challenges, both quantum algorithms and quantum architectures analogous to classical machine learning models have already been developed. One such architecture, and the core of this study, is Quantum Convolutional Neural Networks (QCNNs). Part of this study builds on previous work and focuses on the effectiveness of these architectures, proposing the use of new circuits and conducting a comparison with classical alternatives, such as convolutional neural networks (CNNs).

These types of models aim to address one of the main problems of circuit training, present solely in the quantum variant, commonly known as barren plateaus. This problem can greatly hinder effective parameter training due to the "flat landscape" that cost functions in these systems may exhibit. Moreover, encoding images into quantum states allows for massive data entanglement, enabling the system to extract relationships and features that would not be possible through traditional means.

With these ideas in mind, the main objectives of this work are as follows: to gain a deeper understanding of how quantum machine learning works, interpret the results of proposed experiments, and derive observations and conclusions from these results. Furthermore, the study aims to explore the possibilities offered by the Hierarchical circuit construction library, reproduce the results presented in the relevant literature, and propose improvements in the models. Finally, the work aims to compare the results of different architectures and models to observe the impact of the circuits and the number of parameters on the training and learning of quantum models. It will also compare the effectiveness of classical and quantum machine learning models to demonstrate the potential of quantum models compared to classical ones, focusing on the complexity and trainable parameters of each. Additionally, the study will assess the importance and influence of the data and its characteristics in the training of the models and their results.

In conclusion, this work is structured as follows in the upcoming chapters. Chapter 2 provides an overview of the state of the art, including the NISQ era, QCNNs, and the effectiveness of systems and models in image and audio classification tasks. Chapter 3 presents the majority of the work conducted, detailing the selection of quantum and classical model architectures, as well as the code used for their definition and construction. This chapter also includes the general methodology and the training approach of the models. Chapter 4 covers the various experiments conducted, along with the results obtained and some observations derived from these results. In Chapter 5, the conclusions drawn from the results are presented, along with potential lines for future work.

Chapter 7

# Conclusions and Future Work

## 7.1. Conclusions

As observed in the results presented, the QCNNs proposed and tested in this work are not yet comparable to relatively simple convolutional neural networks (CNNs) as shown by figure 7.1, highlighting the long path ahead in the field of quantum machine learning (QML).

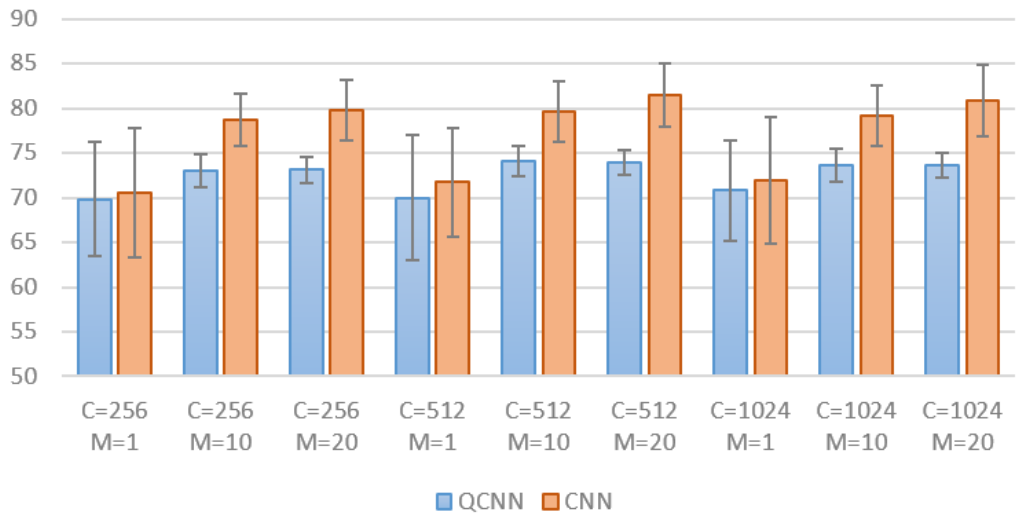


Figure 7.1: Comparison of global test accuracy between QCNNs and CNNs for each type of preprocessing: $C$ represents the number of features for the images and $M$ is the number of samples per audio. Data is provided by tables 4.9 and 4.15.

However, despite these somewhat discouraging results, there are several important points to consider:

**Number of parameters:** The first point to highlight is the size and complexity of each model used in the experiments. While QCNNs and CNNs cannot be canonically compared, the number of trainable parameters is a characteristic they share. Table 7.1 shows the number of parameters for each type of model and circuit tested in this work.

| Features | QCNN 256 | QCNN 512 | QCNN 1024 | CNN 256 | CNN 512 | CNN 1024 |
|---|---|---|---|---|---|---|
| **Trainable parameters** | 51 | **68** | 68 | 8772 | **29572** | 61572 |

Table 7.1: Number of trainable parameters per experiment.

From tables 4.9 and 4.15, it is clear that, overall, the best performance of the proposed quantum models achieved 74.1% accuracy, while the best-performing CNN reached 81.5%. The 7.4% difference between these models may seem relevant at first glance, but the increasing number of parameters in the classic model is also significantly higher: a 33% in the quantum case, but more than 3 times the ammount for the CNN.

**Learning Rate Differences:** Another curious aspect is the small difference in learning rates required for each model, with quantum models needing a smaller rate ($lr = 0.01$) than classical ones ($lr = 0.001$).

This slight difference can be explained appealing to a physical quality that ony appears in quantum models: entanglement. As suggested by [10], logic quantum gates have more influence in the global quantum state even though they don't act in all available qubits. This way, learning rate does not need to be as high, because a small variation in a rotation angle in a specific gate might produce changes in the rest of qubits in the architecture.

**Result Variability and Coefficient of Variation:** By observing tables 4.9 and 4.15, we see that, although CNNs generally perform better, the quantum models have more stable training, as indicated by higher coefficients of variation in most cases. The only exception here is the experiment with 512 features and 1 audio sample, where the quantum model produced a $CV = 0.099$, and the CNN obtained a $CV = 0.088$. This suggests that quantum models may exhibit more stability, providing more robust results than classical alternatives.

**Overfitting:** Finally, it's also important to take into account this phenomenon to verify the viability of proposed models. In this case, quantum models present better trends in learning curves and more stable behaviour between train and test accuracies than classic convolutional nets. This is a very valid point that may determine in a way how training methods would develop for quantum models in the future, with less attention to minimize overfitting.

When comparing CNNs and QCNNs, it becomes evident that QML models still have much to learn, as CNNs outperform QCNNs with larger datasets. However, it is also worth noting that the number of parameters required for CNNs to achieve these results increases considerably with higher image resolution, whereas QCNNs use a much smaller number of parameters. This highlights the potential effectiveness of QML models, particularly for datasets with limited data or features.

## 7.2. Future work

As previously mentioned, QML is still in its early stages of development, with numerous ideas and future research lines yet to be explored. These could range from improving train-

ing times to exploring better preprocessing techniques for representing data in quantum states and testing new models, architectures, and training strategies.

First, there is potential for investigating the offloading of computations to GPUs or other devices to improve performance and reduce wait times. Additionally, the use of multithreading for testing multiple models simultaneously or benchmarking libraries like lazyqml could also be explored. Furthermore, the use of platforms like QIBO for executing quantum circuits on real quantum hardware could shed light on the current effectiveness of these models and ansatzs in non-ideal environments influenced by quantum noise.

Regarding data processing, further exploration into the relationship between the data, preprocessing and models could lead to several alternatives. One potential avenue is studying these models with more datasets related to audio, such as Yaseen or ICBHI, which could contribute to advances in diagnosing heart and lung diseases. Mel spectrograms are one of the simplest ways to visually represent a sound signal, but they are not the only available method. MFCCs are another widely used signal processing technique, and the DCT has been observed to have good data compression capabilities. Exploring the effects of these transformations on data could be interesting, particularly for datasets with limited features or small sample sizes.

The choice of 3-second audio samples (or window length) is relatively arbitrary for fragmenting audio. Studying other sample lengths could provide more information for better subsequent training. All these variables could shed light on a better understanding of data encoding in quantum states and how it relates to model effectiveness, considering the quantity and quality of available training data.

Focusing now on quantum architectures and circuits, the following ideas are proposed. If we want to construct quantum circuits that more accurately represent CNN models, we could extend the circuit with more qubits in the following way (for simplicity, this is shown with only 2 filters):

- Duplicate the number of qubits and build the initial circuit twice side by side. Then, using the qubits that should return the output, extra layers could be added to perform the function of linear or fully connected layers, or a binary tree structure could be used until only one qubit remains for measuring the result. However, this idea is not efficient, as duplicating the number of qubits exponentially increases the memory needed to simulate the circuit.

- Alternatively, quantum computing's power could be leveraged by adding only one qubit, doubling the number of features that can be encoded in a quantum state. With this approach, two copies of the image could be introduced as features, allowing the circuit to learn differently from each copy. Another possibility is to use two contiguous convolutional layers, thereby increasing the number of parameters and the model's expressiveness without increasing the number of qubits.

Finally, investigating hybrid models, i.e. performing convolutions classically but classification in a quantum way, or vice versa, could also be considered.

# Bibliografía


[1] Chen, L.-C., Collins, M., Zhu, Y., Papandreou, G., Zoph, B., Schroff, F., Adam, H. y Shlens, J. Searching for efficient multi-scale architectures for dense image prediction. En *Advances in Neural Information Processing Systems* (editado por S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi y R. Garnett), vol. 31. Curran Associates, Inc., 2018.

[2] Combarro, E. y Gonzalez-Castillo, S. *A Practical Guide to Quantum Machine Learning and Quantum Optimization: Hands-on Approach to Modern Quantum Algorithms*. Packt Publishing, 2023. ISBN 978-1-80461-830-1.

[3] Cong, I., Choi, S. y Lukin, M. D. Quantum convolutional neural networks. *Nature Physics*, vol. 15(12), páginas 1273–1278, 2019. ISSN 1745-2481. Publisher: Nature Publishing Group.

[4] Grant, E., Benedetti, M., Cao, S., Hallam, A., Lockhart, J., Stojevic, V., Green, A. G. y Severini, S. Hierarchical quantum classifiers. *npj Quantum Information*, vol. 4(1), páginas 1–8, 2018. ISSN 2056-6387. Publisher: Nature Publishing Group.

[5] Hur, T., Kim, L. y Park, D. K. Quantum convolutional neural network for classical data classification. *Quantum Machine Intelligence*, vol. 4(1), página 3, 2022. ISSN 2524-4906, 2524-4914. ArXiv:2108.00661 [quant-ph].

[6] Incudini, M., Grossi, M., Ceschini, A., Mandarino, A., Panella, M., Vallecorsa, S. y Windridge, D. Resource Saving via Ensemble Techniques for Quantum Neural Networks. *Quantum Machine Intelligence*, vol. 5(2), página 39, 2023. ISSN 2524-4906, 2524-4914. ArXiv:2303.11283 [quant-ph].

[7] Liao, Y., Hsieh, M.-H. y Ferrie, C. Quantum optimization for training quantum neural networks. *Quantum Machine Intelligence*, vol. 6(1), página 33, 2024. ISSN 2524-4914.

[8] Lourens, M., Sinayskiy, I., Park, D. K., Blank, C. y Petruccione, F. Hierarchical quantum circuit representations for neural architecture search. *npj Quantum Information*, vol. 9(1), página 79, 2023. ISSN 2056-6387.

[9] Pesah, A., Cerezo, M., Wang, S., Volkoff, T., Sornborger, A. T. y Coles, P. J. Absence of Barren Plateaus in Quantum Convolutional Neural Networks. *Physical Review X*, vol. 11(4), página 041011, 2021. Publisher: American Physical Society.

[10] Scala, F., Ceschini, A., Panella, M. y Gerace, D. A General Approach to Dropout in Quantum Neural Networks. 2023.

[11] Schuld, M. y Petruccione, F. *Machine Learning with Quantum Computers*. Quantum Science and Technology. Springer International Publishing, 2021. ISBN 978-3-030-83098-4.

[12] Souto, J., Botella, G., García, D., Murillo, R. y del Barrio, A. Neuromorphic Circuit Simulation with Memristors: Design and Evaluation Using MemTorch for MNIST and CIFAR. 2024. ArXiv:2407.13410 [physics].

[13] Vatan, F. y Williams, C. Optimal quantum circuits for general two-qubit gates. *Physical Review A*, vol. 69(3), página 032315, 2004. Publisher: American Physical Society.

[14] Yanofsky, N. S. y Mannucci, M. A. *Quantum Computing for Computer Scientists*. Cambridge University Press, Cambridge, 2008. ISBN 978-0-521-87996-5.

[15] Zoph, B. y Le, Q. Neural architecture search with reinforcement learning. En *International Conference on Learning Representations*. 2017.