

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Desarrollo de una aplicación móvil que pronostique la alta y baja de acciones comerciales mediante el uso de redes neuronales.

Juan Esteban Dávila Salas

Ingeniería En Ciencias De La Computación

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
Ingeniero en Ciencias de la computación

Quito, 20 de diciembre de 2022

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

**HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA**

Desarrollo de una aplicación móvil que pronostique la alta y baja de acciones comerciales mediante el uso de redes neuronales.

Juan Esteban Dávila Salas

Nombre del profesor, Título académico

Fausto Pasmay, MSc.

Quito, 20 de diciembre de 2022

© DERECHOS DE AUTOR

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en la Ley Orgánica de Educación Superior del Ecuador.

Nombres y apellidos: Juan Esteban Dávila Salas

Código: 00206604

Cédula de identidad: 1719738450

Lugar y fecha: Quito, 20 de diciembre de 2022

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en <http://bit.ly/COPETheses>.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on <http://bit.ly/COPETheses>.

RESUMEN

En la actualidad, una gran y novedosa fuente de riqueza es la compra y venta de acciones comerciales. Esto ha capturado el interés de las personas, pero la poca disponibilidad de recursos y tiempo para aprender sobre el tema es un limitante para la mayoría, lo cual resulta en el abandono de esta actividad. Por este motivo, se planteó como proyecto desarrollar una aplicación móvil que pronostique 30 días al futuro el precio de acciones comerciales ecuatorianas. Para ser más específico de las empresas Corporación favorita, Banco Guayaquil y Cervecería Nacional, ya que en internet se puede encontrar mucha información de empresas internacionales como Apple, Amazon, Facebook, pero muy poca información y más que nada herramientas para acciones ecuatorianas. Para abordar el problema se desarrolló un modelo neuronal recurrente LSTM en Python, el cual es entrenado con los datos de la bolsa de valores de Quito y Guayaquil para luego ser consumido en una aplicación amigable para el usuario desarrollada en Android Studio. En esta se presentará los datos históricos de cada acción utilizando directamente los datos descargados de la bolsa de valores de Quito y, junto a los gráficos de predicción, se los presentará en gráficos que permitan la interacción con el usuario.

Palabras Clave: precio de acciones, predicción, redes neuronales LSTM, aplicación móvil, Android estudio, java, Python.

ABSTRACT

Nowadays, a great new source of wealth is the buying and selling of trading stocks. This has greatly captured the interest of people, but the limited resources and time to learn about the subject is a limitation, which results in the abandonment of this activity. For this reason, this project was proposed to develop a mobile application that forecasts the price of Ecuadorian commercial shares 30 days into the future. To be more specific about the companies “Corporación Favorita”, “Banco Guayaquil” and “Cervecería Nacional”, since on the internet you can find a lot of information about international companies such as Apple, Amazon, Facebook, but very little information and mostly tools for Ecuadorian actions. To address the problem, an LSTM recurrent neural model was developed in Python, which was trained with data from the Quito and Guayaquil stock markets and then consumed in a user-friendly application developed in Android Studio. Here, the historical data of each action was presented, using directly the data downloaded from the Quito stock market and, together with the prediction graphs, they were presented in a way that allowed interaction with the user.

Key words: stocks, prediction, neural networks, mobile application, java, Android studio, artificial intelligence.

TABLA DE CONTENIDO

1. INTRODUCCIÓN.....	10
1.1 Estado del arte.....	10
1.2 Descripción del problema	13
1.3 Objetivo General.....	14
1.4 Objetivo Específico.....	14
2. DESARROLLO DEL TEMA.....	16
2.1 Definición de datos	16
2.1.1 Preprocesamiento de datos.....	16
2.2 Desarrollo aplicación	17
2.2.1 Desarrollo UI.....	17
2.2.2 Requerimientos de la aplicación.	19
2.2.3 Librerías utilizadas.	19
2.2.4 Diagramas de caso de uso.	21
2.2.5 Diagramas UML.....	22
2.3 Desarrollo red neuronal	30
2.3.1 Preprocesamiento de datos.....	30
2.3.2 Funcionamiento del Modelo.....	31
2.3.3 Modelo neuronal recurrente LSTM.....	31
2.3.4 Entrenamiento.	32
2.3.5 Resultados.	33
3. CONCLUSIONES.....	34
3.1 Limitaciones.....	35
3.2 Mejoras Futuras	36
4. REFERENCIAS BIBLIOGRÁFICAS	37
Anexo A: Diagramas UML	39
Anexo B: Manual de Usuario	40
Anexo C: Uso de Modelo Keras en Android.....	42
Anexo D: Código Fuente.....	45

ÍNDICE DE TABLAS

Tabla#1: Resultados de predicción.....	33
--	----

ÍNDICE DE FIGURAS

Figura #1. Diagrama de Red LSTM	11
Figura #2. Arquitectura aplicación MVC	12
Figura #3. Captura de pantalla de la estructura del Excel utilizado como dataset.	16
Figura #4. Diagrama de caso de uso.....	21
Figura #5: Diagrama UML clase acción.....	22
Figura #6: Diagrama UML clase modelo	24
Figura #7: Diagrama UML clases AdapterAcciones y AdapterFavorita	25
Figura #8: Diagrama UML clases MainActivity y MainActivityFragment	26
Figura #9: Diagrama UML clases DownloadFileTask.....	27
Figura #10: Diagrama UML clases Statics Activity y StaticsFragment.....	29
Figura #11: Estructura de los datos para funcionamiento de la red.....	31
Figura #12: Gráfico Loss vs Epochs de cada modelo.	32
Figura #13: Comparación datos reales con datos de prediccion	33
Figura #14: Diagrama UML de aplicación.....	39
Figura #15: Pantalla de inicio antes y después de carga de datos.	40
Figura #16: Estructura de la pantalla Estadísticas	41
Figura #17: Flujo de pantallas para agregar una acción como favorita.....	41
Figura #18: Comparación resultados Tensorflow Lite con modelo Keras	42
Figura #19: Código que demuestra el uso del modelo “tsflite” en Android Studio	43

INTRODUCCIÓN

Estado del arte

Las acciones de una empresa son aquellas partes en la que el capital de una empresa se divide, por lo que al comprar una acción nos estamos volviendo acreedores de una fracción de la empresa. El precio de una acción se ve definido por una oferta pública de venta [1] en el que el precio se ve definido por la situación del mercado con respecto a la oferta y demanda hacia la empresa en cuestión. Estos precios son super bursátiles, tienden a variar diariamente, y no solo se ven afectados por el capital de la empresa, sino también por factores sociales, por ejemplo, cuando la empresa cambia de directiva o se ve envuelta en un escándalo, los precios de acciones tienden a bajar; por otro lado; si existe mucha demanda hacia las acciones, pero poca oferta, estas tienden a subir.

Cuando una persona se hace acreedora de una acción, según la empresa, tiene derecho o no a ser acreedor de un dividendo periódico. Un dividendo vendría a representar que tanto porcentaje de una acción o que beneficio económico recibe una persona por ser acreedor de una acción [2]. Otra fuente de ingresos es el trading. El trading es la compraventa de acciones cotizadas, en donde se busca obtener un beneficio económico por la plusvalía que la acción genere [3]. Es decir que cuando una acción que se compra a precio bajo y se vende a precio alto, el porcentaje de plusvalía que represento la acción en ese periodo de compraventa son las ganancias para el acreedor de la operación. Por esta razón, los accionistas siempre están pendientes de los gráficos de tendencia y el historial de una acción, para lograr determinar si la compraventa de una acción va a salir rentable en un periodo de tiempo determinado.

A partir de esto, se ha planteado utilizar redes neuronales recurrentes LSTM (Long short-term memory) para poder predecir el precio de acciones comerciales mediante el estudio de datos históricos. La característica principal de este tipo de redes es que la información puede persistir en el diagrama de red [4]. Básicamente recuerda estados previos para

utilizar esta información y mediante probabilidades y cálculos predecir el siguiente valor. Las redes LSTM son capaces de capturar patrones a lo largo del tiempo y determinar que patrón o que información es importante para aumentar la precisión de la predicción. Las redes LSTM están en capacidad de añadir o eliminar la información que consideren relevante para el procesamiento de la secuencia.

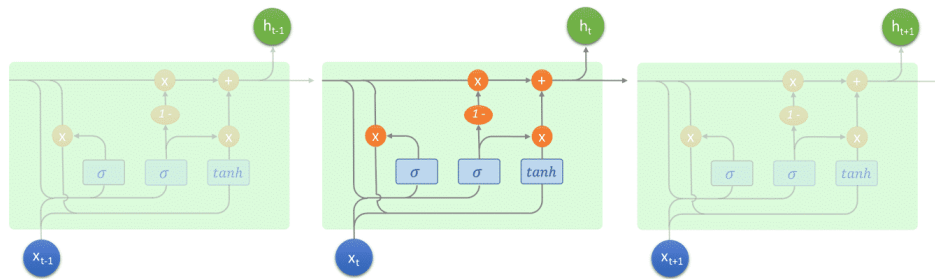


Figura #1. Diagrama de Red LSTM [5]

El funcionamiento de esta se basa en celdas que contienen la estructura de tres capas, las cuales son redes neuronales normales con su respectiva función de activación. Estas gates son:

- Capa de olvido: que decide qué información se queda, indicando valor 0 cuando no es relevante y 1 cuando sí.
- Capa de entrada: se encarga de actualizar el estado de celda logrando memoria a largo plazo con la función de activación Tangh garantizando que los valores permanezcan entre un número positivo (1) y negativo (-1), lo que permite aumentar o disminuir el valor del estado de la celda.
- Capa de salida: que es usada para actualizar el estado oculto de la celda que va a ser utilizada por la siguiente para poder determinar que se olvida y que se mantiene. Además de definir qué información es importante o no.

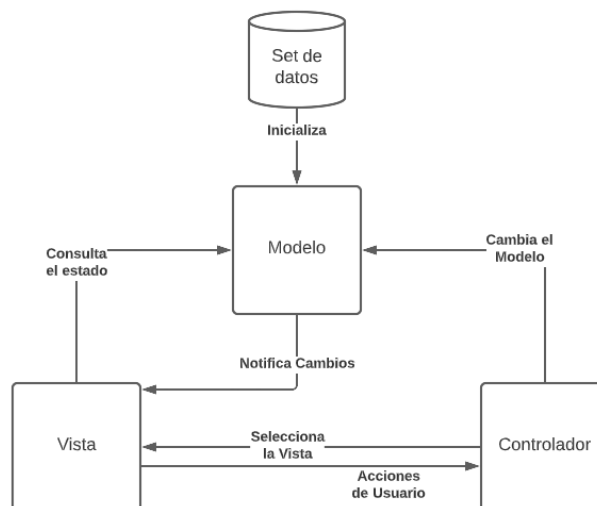


Figura #2. Arquitectura aplicación MVC

Para la arquitectura de la aplicación se presenta una modelo vista controlador en donde se separa la vista de los datos y estas solo pueden interactuar mediante un controlador. El modelo es inicializado tras la descarga del set de datos en el internet, este notifica un cambio a la vista para que sea actualizada con los respectivos datos.

- Vista: vendría a ser todos los layouts desarrollados en Android Studio, donde se presentan los datos. En estos el modelo notificara si existe un cambio en los datos para que estos se actualicen de manera correspondiente.
- Modelo: El modelo esta desarrollado en un patron de diseño singleton el cual estipula la creación de un método estático dentro de sí misma para garantizar que solo exista una instancia de este tipo y de esta manera proporcionar un único acceso a este para cualquier otra clase. Este solo puede ser cambiado por las clases que corresponden al controlador.
- Controlador: Se encarga de asignar a cada vista su respectivo elemento java, para luego ser programado y manipulado. Según la interacción del usuario, se llama a los datos de la clase modelo para poder ser presentado en la correspondiente vista. Si el usuario realiza un cambio, como es el caso de

agregar acciones favoritas, el controlador se encarga de crear la instancia y almacenarla en la respectiva variable del modelo.

La interacción como tal vista modelo no existe, el modelo solo notifica a la vista si ha existido un cambio en los datos para que esta se mantenga actualizada.

Además, para el desarrollo tanto de la red neuronal como de la aplicación móvil, se ha basado en los lineamientos de desarrollo IEEE. Para el desarrollo de la aplicación se ha basado en el ISO/IEC/IEEE 29148-2018 [6] en donde se estipula los lineamientos para el buen desarrollo de software a lo largo del ciclo de vida de esta. En base a esta, se ha desarrollado las respectivas clases con sus respectivos atributos según la norma [6]. Para el desarrollo de la red neuronal se ha basado en la norma ISO/IEC 23053:2022 en donde se estipula el desarrollo de sistemas que utilizan machine learning. En esta se define que, para modelos de regresión, las variables de entrada y de salida deben ser continuas, se define una norma para la retro propagación en donde el output de la red neuronal LSTM optimiza los pesos mediante la propagación del error a través de la red. [7]

Descripción del problema

Al existir un mercado que busca una predicción de los precios de las acciones, o un análisis gráfico de la tendencia de un precio para así tomar decisiones sobre la compraventa de acciones, se ha planteado desarrollar una red Neuronal LSTM que mire 60 días al pasado para predecir 30 días al futuro. Estos datos obtenidos por la inteligencia artificial van a ser desplegados en una aplicación móvil junto a LineCharts que permitan al usuario analizar y entender cuál es los patrones que está tomando el precio de una acción. De esta manera el accionista puede tomar decisiones certeras que le permitan generar una ganancia económica.

Este tipo de aplicaciones existen en el mercado, pero se enfocan mucho en la compraventa de criptomonedas o del precio de empresas internacionales grandes, generando una

limitación para los ecuatorianos ya que, como tal, no existe una aplicación, o incluso, una herramienta web que permita visualizar y analizar de manera rápida y eficiente los precios y movimientos de las acciones ecuatorianas. Actualmente, los ecuatorianos para poder generar ganancia de acciones nacionales deben recurrir a bancos o a corredores de bolsa, confiando su dinero a desconocidos.

A partir de esto se han planteado los siguientes objetivos:

Objetivo General

Desarrollar una aplicación en Androide que tenga la capacidad de predecir la alta o baja del valor de las acciones de las empresas “Corporación Favorita C.A”, “Banco Guayaquil S.A” y “Cervecería Nacional CN SA” mediante el uso de redes neuronales.

Objetivo Específico

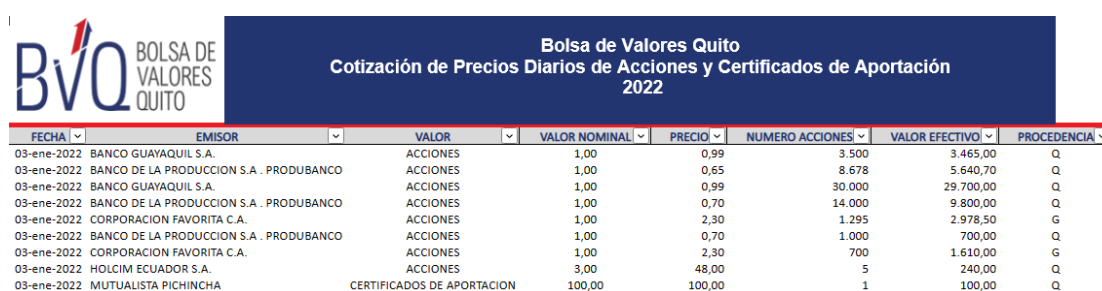
- Manejar un dataset predefinido a partir del cual se realizará la carga de conjuntos de datos en el aplicativo móvil.
- Definir las variables financieras adecuadas que funcionen como la base del desarrollo y proceso de aprendizaje de la red neuronal.
- Desarrollar un modelo neuronal capaz de proyectar futuros valores de acciones de compañías ecuatorianas en el entorno de desarrollo Android Studio.
- Interpretar los resultados obtenidos por el modelo y presentarlos en gráficos de tendencia de la evolución de los precios.
- Crear un aplicativo móvil que muestre datos de acciones comerciales ecuatorianas históricas y futuras, en una interfaz gráfica de fácil uso para el usuario, cumpliendo con los estándares UX/UI.
- Realizar un menú que permita al usuario la selección de las empresas que desea visualizar.

- Permitir al usuario seleccionar el rango de tiempo entre 1 día, 1 semana y 1 mes para la proyección futura del tiempo.
- Permitir al usuario visualizar datos pasados de las acciones en un rango de tiempo determinado.
- Devolver un valor del porcentaje de subida o bajada de la acción en el rango de tiempo determinado
- Almacenar datos históricos con un máximo de un año y eliminar valores previos a esto.
- Generar reporte de las acciones en el que se visualice promedio, desviación estándar, variación, precio inicial, precio, precio actual, BPA, P/E, Porcentaje Diario y Porcentaje Anual.

DESARROLLO DEL TEMA

Definición de datos

Para poder cumplir con los objetivos se va a utilizar los datos obtenidos de la bolsa de valores de Quito [8] en la cual se puede encontrar bastante información y diferentes boletines de acciones comerciales ecuatorianas. Sin embargo, la que más destaca y es útil para el desarrollo del proyecto es el boletín de cotizaciones históricas - acciones, el cual contiene el precio de cada transacción realizada dentro de la bolsa de valores de Quito y de Guayaquil¹



FECHA	EMISOR	VALOR	VALOR NOMINAL	PRECIO	NUMERO ACCIONES	VALOR EFECTIVO	PROCEDENCIA
03-ene-2022	BANCO GUAYAQUIL S.A.	ACCIONES	1,00	0,99	3.500	3.465,00	Q
03-ene-2022	BANCO DE LA PRODUCCION S.A. - PRODUBANCO	ACCIONES	1,00	0,65	8.678	5.640,70	Q
03-ene-2022	BANCO GUAYAQUIL S.A.	ACCIONES	1,00	0,99	30.000	29.700,00	Q
03-ene-2022	BANCO DE LA PRODUCCION S.A. - PRODUBANCO	ACCIONES	1,00	0,70	14.000	9.800,00	Q
03-ene-2022	CORPORACION FAVORITA C.A.	ACCIONES	1,00	2,30	1.295	2.978,50	G
03-ene-2022	BANCO DE LA PRODUCCION S.A. - PRODUBANCO	ACCIONES	1,00	0,70	1.000	700,00	Q
03-ene-2022	CORPORACION FAVORITA C.A.	ACCIONES	1,00	2,30	700	1.610,00	G
03-ene-2022	HOLCIM ECUADOR S.A.	ACCIONES	3,00	48,00	5	240,00	Q
03-ene-2022	MUTUALISTA PICHINCHA	CERTIFICADOS DE APORTACION	100,00	100,00	1	100,00	Q

Figura #3. Captura de pantalla de la estructura del Excel utilizado como dataset.

Como se puede apreciar en la Imagen 2, se puede observar que el dataset nos devuelve el nombre del Emisor, fecha de cuando se realiza la transacción, precio de la acción, los cuales serán nuestras variables más importantes a lo largo del proyecto. Además, este dataset consta con una actualización de los datos desde el año 2021 hasta la actualidad.

Preprocesamiento de datos.

Como se mencionó, para cumplir nuestro objetivo debemos seleccionar solo aquellas acciones de nuestro interés. Para esto se realiza un preprocesamiento de los datos tanto en Android como en Python para el uso de la aplicación y el entrenamiento de la Red Neuronal.

¹ Link de descarga del dataset: <https://www.bolsadequito.com/uploads/estadisticas/boletines/cotizaciones-historicas/acciones.xls>

Datos en Red Neuronal

Para el entrenamiento de la red neuronal se separó en 3 diccionarios de la librería pandas, uno por cada acción, en donde se eliminó las columnas innecesarias dejando solo "Fecha" "Emisor" y "Precio". Debemos asegurarnos de que los datos estén en orden de fecha y realizamos una función de normalización obtenida por sklearn [9].

Datos en Android Studio.

Los datos al ser descargados mediante un enlace directo, se realiza una descarga con Download Manager, él cual es un servicio del sistema que maneja descargas HTTP, y de manera asíncrona dirigida a los archivos privados de la aplicación [10]. De esta manera se puede realizar la descarga del archivo mediante el URL de este a la web. Esta descarga se realiza solo si la aplicación tiene internet, caso contrario utiliza los datos previamente descargados. Para el preprocesamiento rápido de los datos en Android Studio, al ser un archivo Microsoft Excel, utilizamos la librería Apache POI [11] la cual nos permite la rápida lectura de las columnas y filas del excel. Se empieza a realizar la lectura recorriendo cada columna, obteniendo solo aquellas Acciones de nuestro interés y asegurándonos que las fechas no se repitan. Almacenamos esto en un Hashmap para luego crear la respectiva clase y almacenarla en la lista del modelo singleton.

Desarrollo aplicación

Desarrollo UI.

Para el desarrollo de la interfaz gráfica de la aplicación, se empezó realizando una investigación, buscando referencias de diseño y maquetas de aplicaciones similares como inspiración y además siempre teniendo presente los objetivos que esta debe cumplir. Primero se buscó una paleta de colores optando por una paleta de colores azules. Definimos una sección de acciones favoritas llamativa donde el usuario pueda accederlas

de manera más rápida a sus acciones preferidas y una sección donde se encuentran todo el mercado de empresas que nuestra aplicación presenta.

Luego, se desarrolló la pantalla de Estadísticas (Figura #16), en donde podemos observar los botones de tiempo, tanto del pasado y futuro como el rango de tiempo. El line chart donde se presentan los datos de la acción y finalmente los datos informativos de cada acción. Una vez realizado este maquetado, se empezó el desarrollo de la aplicación en Android. Con la maqueta ya definido facilitamos mucho el proceso de desarrollo UI en Android Studio porque como ya tenemos claro cómo se va a ver la aplicación, solo debemos seleccionar los componentes correctos. Para el diseño de la primera pantalla (Figura #14) se utilizó dos recycler views los cuales nos permiten replicar elementos a partir de los valores de una lista. Estos se relacionan con un respectivo adaptador, el cual se encarga de obtener los datos del modelo y los relaciona con su respectiva vista. En la segunda pantalla (Figura #15) se utilizó dos RadioGroups los cuales nos ayudan a simular un botón de switch. Este nos ayuda a conocer el estado de todos los botones sin tener que programarlos uno por uno. Finalmente, para el desarrollo del Line Chart, se utilizó una librería grafica llamada MPAndroidChart [12].

Para el desarrollo de este gráfico simplemente se requiere un Entry set que vendría a ser nuestra lista de precios y las fechas. Configuramos el gráfico para que sea un Line Chart y que se mantenga al diseño de nuestra aplicación, configurando los colores de la línea, donde van situados las etiquetas, como se visualizan las líneas de fondo, etc.

Finalmente, para buscar que la aplicación tenga un mayor alcance, se utilizó el patrón de diseño a Maestro detalle en donde las actividades se muestran en fragmentos. Esto es de bastante ayuda para que la aplicación tenga diferentes formas según el tamaño de pantalla desplegado. Se realizó el desarrollo del diseño y la respectiva programación para layouts como tablets.

Requerimientos de la aplicación.

La aplicación para poder ser ejecutada en un dispositivo móvil debe requerir con un nivel de API mayor a 26, es decir, tener una versión de Android igual o mayor a 8. Este nivel de Api permite la ejecución en 90% [13] de dispositivos actualmente activos en Android Studio. La aplicación requiere de 48.79 MB para poder ser instalada y finalmente de conexión a internet para poder ser ejecutada por primera vez. A partir de esta primera ejecución ya no es necesario una conexión.

Librerías utilizadas.

TensorFlowLite [14]: Para el desarrollo de la red neuronal se utilizó la librería Keras, la cual es una librería high-level que corre sobre TensorFlow [15]. A partir de esta librería nos permite exportar nuestro modelo y utilizarlo en diferentes módulos que el mismo tensorflow admite, como es el caso de tensorflow Lite. La librería Tensorflow Lite es una herramienta de tensorflow que nos permite utilizar modelos entrenados en aplicativos móviles y permite implementar el aprendizaje automático dentro del mismo [11]. Para el funcionamiento de esta se debe convertir el modelo en tflite para poder ser utilizado por la librería. Una vez realizado esto se procede a la preparación del entorno en donde se debe agregar las dependencias de TensorFlow Lite a “build.gradle” del proyecto android:

- “implementation 'org.tensorflow:tensorflow-lite-task-vision-play-services:0.4.2”
- “implementation 'com.google.android.gms:play-services-tflite-gpu:16.1.0”

Con esto podemos utilizar las herramientas incorporadas a Android Studio como es la de incorporar nuestro modelo “.tsflite” de manera automática a la carpeta assets/ml.

Apache POI: Librería propia de java JDK 8 que se puede utilizar en nuestro proyecto Android al ser está programada en java. Esta librería tiene como objetivo la manipulación de archivos Office Open XML, es decir archivos Excel microsoft office [11] Esta librería

nos permite la lectura de los datos de manera óptima para poder desplegar los datos sin la necesidad de crear una base de datos. Para agregar esta librería a Android se debe agregar las siguientes dependencias:

- “implementation 'org.apache.poi:poi:5.2.2'”
- “implementation 'org.apache.poi:poi-ooxml:5.2.2'”

MPAndroidChart [12]: Esta librería grafica nos permite la visualización de los datos previamente leídos y almacenados en el respectivo gráfico LineChart. Es una herramienta de libre uso desarrollada por Philipp Jahoda la cual nos permite, mediante un Entryset de datos, crear un linechart, además, con su amplia libertad al momento de configurar los componentes, podemos cumplir los objetivos de diseño y funcionalidad de nuestra aplicación. Para agregar la librería se debe agregar la siguiente dependencia:

- “implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'” [12]

Diagramas de caso de uso.

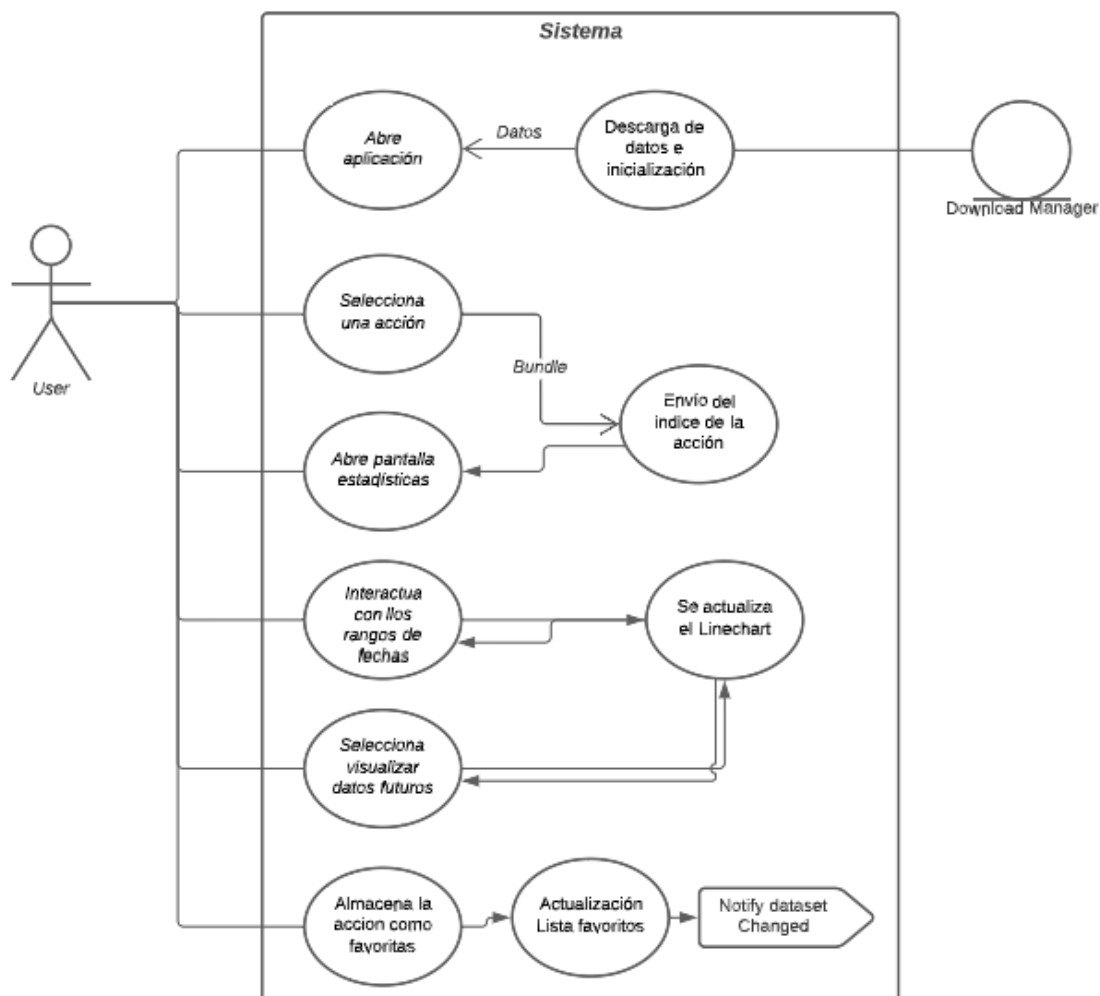


Figura #4. Diagrama de caso de uso.

El usuario ingresa a la aplicación en donde debe esperar a que los datos se descarguen siendo la primera vez que ingresa o se actualicen, se despliega las listas en sus respectivas posiciones, si el usuario tiene acciones favoritas puede seleccionar de esta lista favoritas, caso contrario las selecciona de la lista mercado. Se le direcciona a la pantalla estadísticas en donde se despliega la información de la acción seleccionada. En esta el usuario interactúa con los botones de tiempo en donde puede seleccionar el rango de visualización del gráfico. Selecciona una fecha que desea visualizar los precios. Cambia el modo de visualización del gráfico para visualizar los datos a futuro y selecciona el rango de fecha a futuro que desea visualizar. Finalmente, el usuario, si es de su agrado e interés los datos

presentados de la acción, la agrega a favoritos y cierra la aplicación. Caso contrario cierra la aplicación.

Diagramas UML.



Figura #5: Diagrama UML clase acción

Clase acción.

Esta clase contiene todas las variables necesarias para poder utilizar el objeto dentro de la interfaz gráfica tanto en la actividad main como en la actividad estadística. Dentro de las que se pueden destacar el boolean `altbaj` nos indica si una acción con respecto al mes anterior está en crecimiento o decrecimiento. Las variables que corresponden a los datos estadísticos y el respectivo `MinFeature` y `scale` [10] de cada acción para la futura desnormalización de los datos. Contiene tres listas importantes, las fechas, el precio, y el bloque de 60 datos normalizados que se van a utilizar para la lectura de nuestra red neuronal. Las funciones, además de cada `set` y `get` de cada variable, tenemos las funciones de cálculo de cada dato estadístico. Dentro de los variables se agrega de manera manual el `ticker` y el `bpa` [17]

- `calculoAnual()`: Se calcula el rango de crecimiento de la acción mediante el uso de la siguiente fórmula:

$$\% = \frac{\text{precioActual}}{\text{precioInicial}_{\text{Anual}}} - 1 * 100$$

- `calculoDiario()`: Se calcula el rango de crecimiento con respecto a un día al pasado:

$$\% = \frac{\text{precioActual}}{\text{precioInicial}_{\text{Dia}}} - 1 * 100$$

- `CalculoMedia()`: función que calcula el promedio de la lista precios mediante el uso de la función `stream average`.
- `calculoSTD()`: Se obtiene el cálculo de la desviación estándar mediante el uso de un bucle, que recorre cada precio restando el promedio, elevando todo al cuadrado. Esto se almacena en una variable para luego ser almacenada en la respectiva variable.

- `calculoVarianza()`: Se calcula la varianza realizando la raíz cuadrada con la función `Math.sqrt` de la desviación estándar.
- `calculoRatio()`: Calcula el porcentaje de beneficio de la acción dividiendo el precio actual con el porcentaje de dividendo de la acción (BPA).
- `inicializarDatos()`: Esta función mediante el uso de bucles, realiza una normalización con respecto a la lista precios para luego retornar los últimos 60 datos siendo este el bloque `inputFeature` para nuestro modelo.

Estas funciones se llaman en el constructor, para que cuando el objeto sea inicializado, se tenga listas las respectivas variables para su uso.

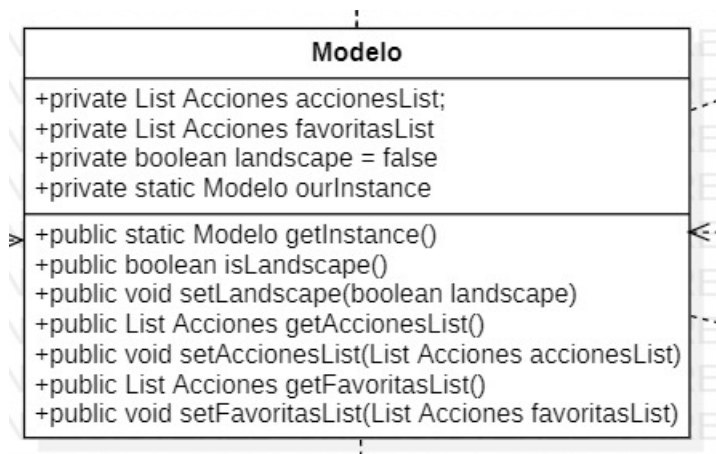


Figura #6: Diagrama UML clase modelo

Clase Modelo.

Esta clase es un singleton el cual, es un patrón de diseño en donde se tiene una clase que se inicializa a sí misma para que esta pueda mantener los datos a lo largo de la ejecución entre actividades. Dentro de esta consta dos listas del tipo acción, `accionesList` y `FavoritasList`. A cada lista se le agrega datos, cuando corresponde, a lo largo de la ejecución de la aplicación. Se tiene una variable `landscape` para determinar si la aplicación se encuentra ejecutándose en dispositivos con pantalla ancha para poder presentar el respectivo UI. Tenemos la variable `ourInstance` la cual es un objeto del tipo `Modelo` para

así crearse dentro de sí misma. Esta clase no tiene constructor, solo la función getInstance, la cual nos va a permitir llamar a la instancia del modelo en las diferentes clases que lo requieran.

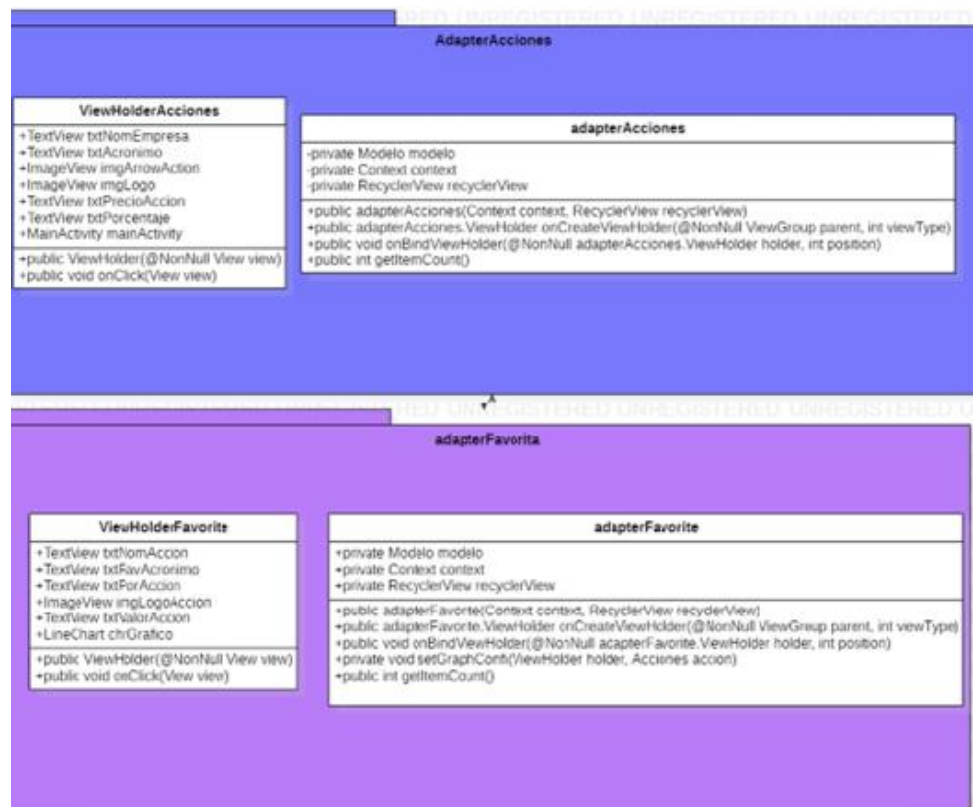


Figura #7: Diagrama UML clases AdapterAcciones y AdapterFavorita

AdapterAcciones y AdapterFavorita

Esta clase nos permite la inicialización del recycler View rcvAcciones y rcvFavoritas dentro de main_activity. En esta se inicializa el modelo de datos para obtener la lista de las acciones y favoritas. Y se realiza la respectiva programación dentro de las funciones preestablecida de la librería que esta solicita para inicializar cada Cardview dentro de la misma. Dentro del adaptadorFavorita se agrega una función la cual inicializa el gráfico con MPAandroid chart en donde solo se despliega los últimos 60 datos, se elimina las funcionalidades de click dentro del gráfico ya que el objetivo es simplemente mostrar un

resumen de los datos en el gráfico, mas no la interacción. A cada cardView se le agrega un onClick en el cual envía un Bundle con los datos de la acción para que estos puedan ser desplegados de manera correcta en statics_activity.

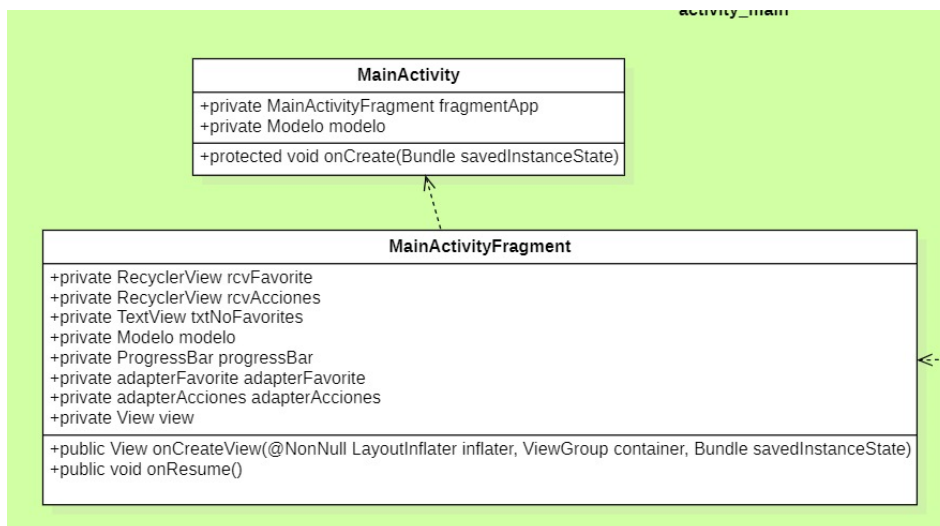


Figura #8: Diagrama UML clases MainActivity y MainActivityFragment

MainActivity y MainActivityFragment

En la clase MainActivity se realiza la respectiva asignación al correspondiente fragmento frgApp y si dentro de la actividad_main se encuentra el fragmento de Estadísticas, se cambia la variable landscape a true, para indicar a toda la aplicación que esta se encuentra en landscape.

Dentro de la clase MainActivityFragment se encuentra toda la inicialización de las vistas asignando las respectivas variables junto a sus respectivos ids. Dentro del onCreateView se realiza la correspondiente llamada a la clase downloadFileTask la cual se encarga de inicializar los datos. De esta manera la actividad se crea y se presenta al usuario sin la necesidad de esperar a que los datos estén inicializados. onResume notifica al adaptador de la lista de favoritas, que los datos han sido cambiados. Si la lista favorita presenta datos, el texto "No existen Favoritos" es eliminado.

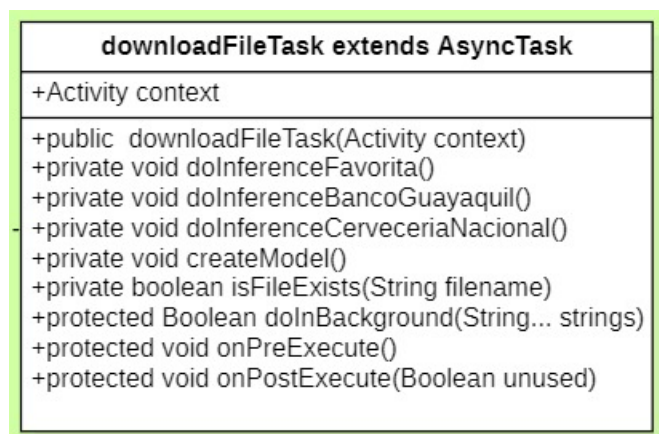


Figura #9: Diagrama UML clases DownloadFileTask

DownloadFileTask

Esta clase se encarga, de manera asíncrona, de la descarga, inicialización de los datos y consumo de los modelos neuronales. Se pasa el contexto de la actividad para poder tener control sobre los assets, ya que los necesitamos para poder llamar a los modelos “. tsflite” y para acceder al almacenamiento interno de la aplicación.

`doInBackground()`: Esta función se encarga de descargar los datos. Primero revisa la conexión a internet, si esta existe se procede a la descarga mediante el uso de `DownloadManager`, caso contrario da como concluida la función y se procede al procesamiento de los datos. Se procede a la descarga del archivo, se revisa si el archivo existe para eliminarlo, ya que como tal no existe una función de `rewrite` sobre un archivo existente, de esta manera evitamos que se creen archivos con otro nombre y de manera constante llenando el almacenamiento del usuario. Se realiza la descarga y mediante el uso un objeto `Cursor` se espera a que la descarga finalice.

- `onPreExecute()`: Una vez finalizada la descarga del archivo, se procede al procesamiento de los datos, en donde se llama a las funciones `createModel()`, y las funciones `doInference()` en donde se realiza el consumo del respectivo modelo por acción. Además, se elimina la visibilidad del objeto `progressBar`.

- `createModel()`: en esta función se realiza el uso de la librería Apache POI para poder realizar la lectura del archivo, se utiliza las variables recomendadas por la librería para poder realizar la respectiva lectura por hojas, columnas y filas. Primero, se define que lea las dos hojas del documento, ya que el documento divide los años por hojas. En cada hoja, los datos de las acciones empiezan en la fila 10 y nuestras columnas de importancia son 1, 2 y 5 en donde se encuentran las fechas, nombres y precios de cada acción. Se realiza el ciclo en donde los datos se van almacenados en un hashmap que contiene como key los nombres de Banco Guayaquil, Corporación Favorita y cervecería nacional. Si el valor de la columna nombre está dentro de estas llaves, se agrega los correspondientes datos. Caso contrario se procede a la siguiente fila. Una vez almacenado solo los datos necesarios, se procede a la creación de cada clase y almacenarlos en la lista del modelo.
- `doInference()`: Cada modelo contiene su propia función para consumir el modelo neuronal de la aplicación y que los datos puedan ser almacenados en su correspondiente variable. Se utiliza el código ejemplo que viene integrado dentro de nuestro modelo, el cual se puede visualizar gracias a la librería TensorFlow Lite. En esta se ingresa al `tensorBuffer` el arreglo del bloque de 60 datos que se va a utilizar para realizar la predicción. Finalmente, los datos son almacenados en un arreglo de `output` para proceder, con el uso del `MinFeature` y el `scaler`, a realizar una normalización inversa de los datos y son almacenados en la lista `PreciosFuturos`.

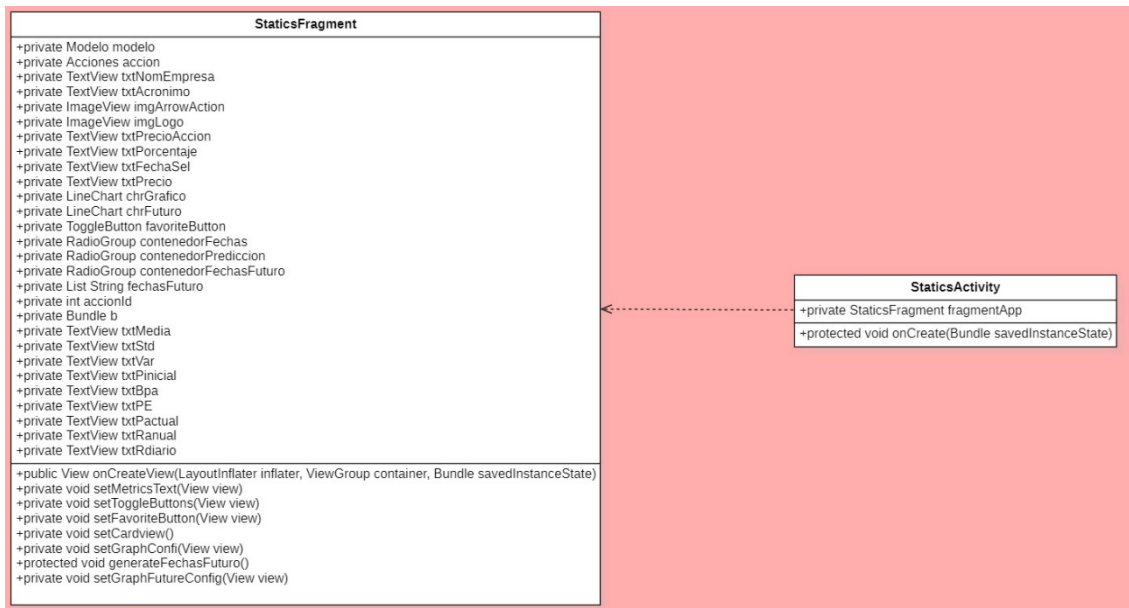


Figura #10: Diagrama UML clases Statics Activity y StaticsFragment

Statics Activity y StaticsFragment

Dentro de la clase statics Activity se inicializa el fragmento de estadísticas. Dentro del fragmento se realiza la correspondiente asignación de cada variable con su respectiva vista. En la función onCreate se llama al bundle para poder obtener el índice de la acción seleccionada. Con este índice se llama a la función del modelo getList().get() para obtener la acción y almacenarla. Dentro del onCreateView se inicializa y se llama a las respectivas funciones para inicializar cada vista como el cardView y ToggleButtons. Además, en esta pantalla se inicializa cada Linechart que corresponde a los datos del pasado y los datos del futuro.

- setGraphConfi(): En esta se inicializa el gráfico mpandroidchart. El gráfico requiere de un ArrayList objeto Entry para las Xs y Ys. Esto se le hace agregando las respectivas listas del modelo de fechas y de precios a la lista Entry. Este entry luego se le agrega al objeto LineChart chrGrafico. Luego se verifica si el booleano isaltbaj de la acción es verdadero o falso para definir los colores sea rojo o verde. Se le agrega un ValueSelectedListener para que el gráfico pueda devolver los

datos seleccionados, tanto de fecha como precio, y se cambie el texto a txtFechaSel y txtPrecio. Se le agrega dentro de la misma un changeListener para que al seleccionar un cambio de fecha, el gráfico sepa a que escalar debe cambiar. Finalmente, se va definiendo y variando las variables de diseño del gráfico para cumplir con los objetivos de diseño.

- setGraphFutureConfig(): Dentro de esta función se inicializa de igual manera al gráfico chrFuturo. A diferencia del anterior gráfico, los datos localDate de fecha deben ser creados. Estos son inicializados mediante la función generateFechasFuturo() que genera una lista de 30 fechas al futuro a partir del día siguiente de ejecución; para luego ser introducidos en los datos del precio futuro en el respectivo Arraylist de Entry.

Desarrollo red neuronal

El desarrollo de la red neuronal se llevó a cabo en Python, en donde se realiza el respectivo preprocesamiento de datos, creación del modelo, entrenamiento de la red y la respectiva conversión del modelo a “.tslite”.

Preprocesamiento de datos.

Como se mencionó anteriormente, se realiza una limpieza de los datos descargados del Excel. Se divide este en tres diccionarios de la librería de panda en donde se almacena solo aquellos datos de "Corporación Favorita", "Banco Guayaquil" y "Cervecería Nacional" y almacenamos de cada uno su fecha y su precio. Nos aseguramos de que el campo fecha se encuentre en orden y procedemos a la eliminación de esta columna ya que no es necesaria para el entrenamiento.

Realizamos una normalización de los datos de precio normalizándolos en valores de 0-1 para facilitar el entrenamiento de la aplicación. Para esto se utilizó la librería sklearn [10]

la cual nos permite el uso de la herramienta "MinMaxScaler" [10] Esta función se basa en la siguiente formula:

$$Scale = \frac{newMax - newMin}{Max - Min} [10]$$

$$MinFeature = newMin - min * scaler [10]$$

Estas fórmulas debemos tenerlas porque van a ser de utilidad para la normalización y la normalización inversa de los datos en android Studio.

Funcionamiento del Modelo

Lo que hace la red LSTM básicamente mirar una cantidad de valores al pasado y a partir de estos valores predecir los siguientes. Se define que el número de valores que mire al pasado es 60 y a partir de este genere 30 valores. Con esto en mente se define el dataset de nuestras x, el cual es un arreglo que contiene grupos de 60 valores y el dataset Y el cual contiene grupos de 30 que siguen después de los 60 valores. Y siempre va a tener los valores esperados de predicción, mientras X los valores que queremos que estudie para realizar la predicción.

```
X_DATOS
[0]-[N1, N2, N3,...,N60] -> [N61, N62, N63...,N90]
[1]-[N61, N62,...,N120] -> [N121,N122,...,N150]

Y_DATOS
[0] - [N61, N62, N63...,N90]
[1] - [N121, N122,..., N150]
```

Figura #11: Estructura de los datos para funcionamiento de la red.

De esta manera el modelo penaliza a la neurona si los valores predichos no son correctos.

Modelo neuronal recurrente LSTM.

Esta red consta con un input layer, el cual tiene un feature range que mira si los valores están o no normalizados, si no realiza su propia función de normalización. Tiene la primera capa LSTM en donde entra el bloque de 60 de X a las 50 unidades y retorna un

arreglo con los 60 datos más los datos de las 50 celdas. Pasa por un dropout para evitar overfitting, nuevamente otra capa lstm, esto más que nada le hice porque como no tenemos muchos datos como quisiera para el entrenamiento, le hice que vuelva a recorrer los 60 datos con los valores aprendidos de las 50 celdas para que vaya afinando estos valores, Ahora solo regresamos el valor de las celdas, una capa dropout y finalmente una capa densa que, con los valores de las celdas, se realiza la predicción mediante probabilidades de los 30 valores.

Entrenamiento.

Se realizó la respectiva división de los datos train, test y validation con el módulo sklearn de "train_test_split" Con estos datos se realizó el entrenamiento de la red neuronal con 500 épocas y un batch size de 32. En el entrenamiento se obtuvo los siguientes resultados de los 3 modelos desarrollados.

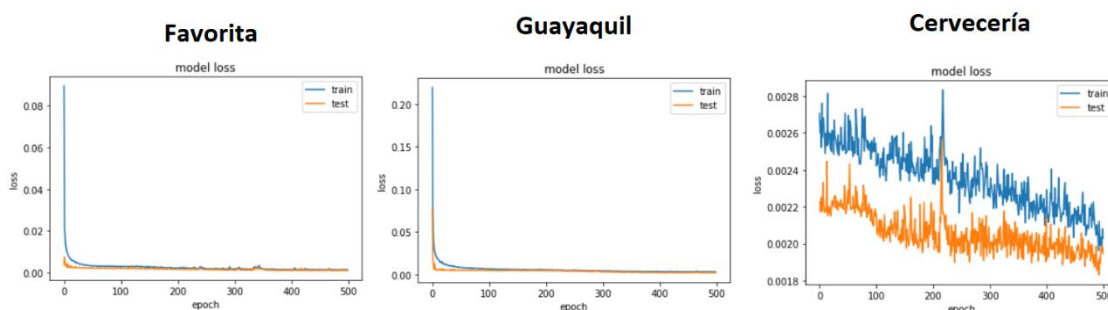


Figura #12: gráfico Loss vs Epochs de cada modelo.

Podemos notar en estos gráficos el funcionamiento de la red, en donde se puede apreciar como en las primeras épocas el modelo predice con un error alto, pero rápidamente se va a afinando hasta que el error es casi inexistente. Podemos observar en el segundo gráfico, que el error entre el train y el test es de 0.001. Por lo que podemos llegar a la conclusión de que nuestro modelo está funcionando y aprendiendo de la manera correcta y realizando buenas predicciones.

Resultados.

	Corporación Favorita	Banco Guayaquil	Cervecería Nacional
Mean Squared Error	0.0013	0.0023	0.0019
Mean Absolute Error	0.021	0.036	0.35
R square	0.95	0.92	0.86

Tabla#1: resultados de predicción

Se utilizó las siguientes métricas para validar una predicción realizada entre datos que el modelo neuronal nunca ha visto y los resultados de los datos reales. Obteniendo buenos resultados de predicción, siendo Cervecería Nacional la que menor resultados presenta dado a que esta acción no presenta tantos datos históricos, por ende, de entrenamiento, como Corporación Favorita y Banco Guayaquil. Se obtuvo los siguientes resultados entre la predicción y los datos reales:

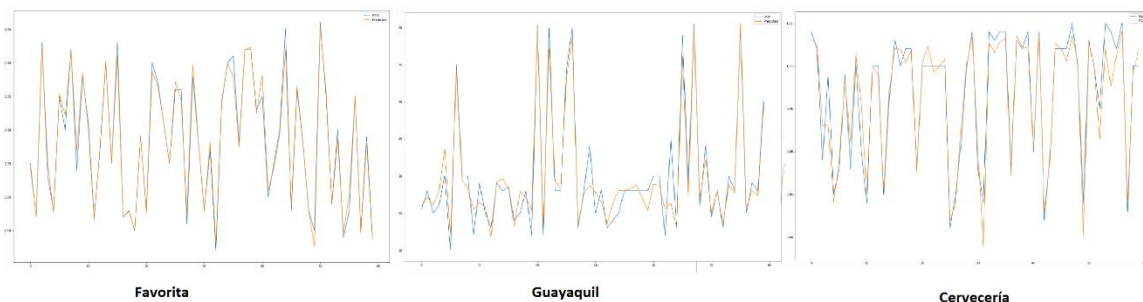


Figura #13: Comparación datos reales con datos de predicción

En la figura número 13 podemos observar en azul los datos reales y en naranja los datos predichos. En estos gráficos podemos destacar que en muchos casos la red neuronal llega a los datos reales, sin embargo, cuando no llega, el modelo se está adelantando a lo que va a suceder con el precio de la acción.

CONCLUSIONES

Los objetivos planteados para el proyecto se cumplieron a complétamente, ya que la aplicación satisface plenamente la necesidad especificada a la cual fue diseñada. La aplicación realiza el respectivo pronosticó de las acciones "Corporación Favorita", "Banco Guayaquil" y "Cervecería Nacional" mediante el uso de un modelo neuronal desarrollado y entrenado en python y consumido en Android Studio.

Tanto dentro de la aplicación como en python se realiza el respectivo manejo de datos. En Android se realiza la carga de datos mediante download mánager, el cual siempre mantiene los datos y las predicciones actualizadas. Estos datos son leídos y almacenados de manera óptima para que se ejecute su respectiva lectura dentro de la aplicación y de los respectivos gráficos que lo requieren. En python se realiza el respectivo preprocesamiento de datos para el respectivo entrenamiento de este obteniendo resultados excelentes.

Se desarrollo un modelo neuronal recurrente lstm capaz de predecir el precio de las acciones a 30 días, mediante el respectivo entendimiento de estas para que esta cumpla las necesidades de nuestra aplicación. Se realiza una visualización al pasado de 60 días para predecir los 30 días al futuro. Este modelo una vez entrenado es exportado y leído en Android Studio.

La aplicación ha sido desarrollada en base al modelo UX/UI realizando una aplicación intuitiva y amigable para el usuario, que no requiere de una curva de aprendizaje para ser utilizada. Además se cumplió con los estándares que solicita Android para que la aplicación pueda ser ejecutada en dispositivos de cualquier tamaño.

Dentro de la aplicación el usuario puede seleccionar las acciones que desea visualizar mediante la carga de los datos en el respectivo RecyclerView junto a su respectivo porcentaje de crecimiento mensuales.

Dentro de la pantalla de cada acción se puede visualizar datos de crecimiento de la acción, entre otros datos de importancia como son: promedio, desviación estándar, variación, precio inicial, precio, precio actual, BPA, P/E, Porcentaje Diario y Porcentaje Anual, los cuales son obtenidos mediante diferentes fórmulas. Los datos históricos se presentan un año al pasado mostrándolos en su respectivo LineChart desarrollado gracias a la librería MPAndroidChart.

En esta pantalla se muestra los datos a futuro de la acción gracias al módulo Tensorflow lite el cual nos permite realizar consumir el modelo desarrollado en python para finalmente desplegar los resultados obtenidos en un linechart que permite al usuario variar los rangos de tiempo que desea visualizar mediante el ajuste del rango del gráfico.

Con esto, llegamos a la conclusión de que la elección de tecnologías fue la correcta ya que la aplicación no tiene problemas para ser ejecutada en dispositivos móviles, es eficiente y no presenta problemas de carga a pesar de estar almacenando y manipulando alrededor de 1185 datos. La red neuronal despliega buenos datos de entrenamiento sin presentar overfitting y con buenos resultados de predicción siendo el mayor 0.95 de Rsquare.

Finalmente, en el plano personal y profesional, el proyecto me permitió explorar a profundidad la funcionalidad de redes neuronales recurrentes y el desarrollo de aplicaciones móviles que tengan inteligencia artificial. El aprendizaje y aplicación de estas tecnologías me proyectará en el futuro en este importante campo que es la inteligencia artificial del conocimiento de la Ingeniería en Sistemas.

Limitaciones

La mayor limitación actualmente en el proyecto es los datos. Los datos dependen íntegramente del mantenimiento y actualización de la bolsa de valores de quito. Estos

datos se llevan actualizando constantemente varios años, pero la eliminación de este correspondería en una mala funcionalidad de la aplicación.

Mejoras Futuras

Como futuras mejoras del aplicativo, se puede agregar mayor funcionalidad e interacción con el usuario con respecto a una acción, como agregar una pantalla de cotizaciones que permita analizar los valores futuros de la acción, en donde el usuario ingrese la cantidad de acciones a comprar y pueda visualizar las ganancias o pérdidas. Además, se podría agregar la funcionalidad de que, dentro de los datos futuros, la aplicación te diga cuando es bueno comprar y vender una acción. Dentro de las limitaciones, se menciona que el dataset, a pesar de ser ideal, puede resultar en problemas a largo plazo, por lo que se debe buscar una mejor fuente de datos que nos permita mantener la integridad de estos. La aplicación ha sido desarrollada con la finalidad de que en un futuro se agregue más empresas ecuatorianas para que el usuario tenga un mayor catálogo de visualización de datos y de valores de predicción según la acción seleccionada.

REFERENCIAS BIBLIOGRÁFICAS

- [1] IG. "Precio de las acciones (definición)". IG. <http://bit.ly/3BOUysy>
- [2] IG. "Qué son los dividendos". IG. <https://bit.ly/3jgL8jg>
- [3] M. Coca. "¿Qué es el 'trading'?" BBVA NOTICIAS. <https://www.bbva.com/es/que-es-trading-que-hace-falta-para-operar/>
- [4] J. I. Garzón. "Cómo usar redes neuronales (LSTM) en la predicción de averías en las máquinas". Blog: GFT. <https://bit.ly/3HQPSpV>
- [5] D. Calvo. "Red Neuronal Recurrente - RNN". Diego Calvo. <https://www.diegocalvo.es/red-neuronal-recurrente/>.
- [6] "ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering," in *ISO/IEC/IEEE 29148:2018(E)*, vol., no., pp.1-104, 30 Nov. 2018, doi: 10.1109/IEEESTD.2018.8559686.
- [7] "ISO/IEC 23053:2022 Framework for Artificial Intelligence (AI) Systems Using Machine Learning (ML)" in *ISO/IEC JTC 1/SC 42*, vol., pp.1, 36 May. 2022.
- [8] "Bolsa de Valores de Quito Sociedad Anónima | La mejor opción de Inversión y Financiamiento del Ecuador". Bolsa de Valores de Quito. <https://www.bolsadequito.com/>.
- [9] L. Buitinck et al., "API design for machine learning software: experiences from the scikit-learn project", *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.
- [10] Android Developers. "DownloadManager Android Developers". Android Developers. <https://developer.android.com/reference/android/app/DownloadManager>
- [11] Apache Software Foundation. "Apache POI - the Java API for Microsoft Documents". Apache POI. <https://poi.apache.org/>.
- [12] P. Jahoda. "GitHub - PhilJay/MPAndroidChart: A powerful 🚀 Android chart view / graph view library, supporting line- bar- pie- radar- bubble- and candlestick charts as well as scaling, panning and animations." GitHub. <https://github.com/PhilJay/MPAndroidChart>.
- [13] Android Developers. "Funciones y API de Android 8.0 Desarrolladores de Android Android Developers". Android Developers. <https://developer.android.com/about/versions/oreo/android-8.0?hl=es-419>.
- [14] TensorFlow. "TensorFlow Lite | ML for Mobile and Edge Devices". TensorFlow. <https://www.tensorflow.org/lite>.
- [15] J. Terra. "Pytorch Vs Tensorflow Vs Keras: Here are the Difference You Should Know". *Simplilearn*. <https://bit.ly/3VbTTIy>
- [16] TensorFlow. "TensorFlow Lite". TensorFlow. <https://www.tensorflow.org/lite/guide>.

[17] MERCAPITAL. "Boletín Acciones". MERCAPITAL. <https://www.mercapital.ec/wp-content/uploads/2022/03/BOLETIN-ACCIONES-AL-28-03-2022.pdf>.

Anexo A: Diagramas UML

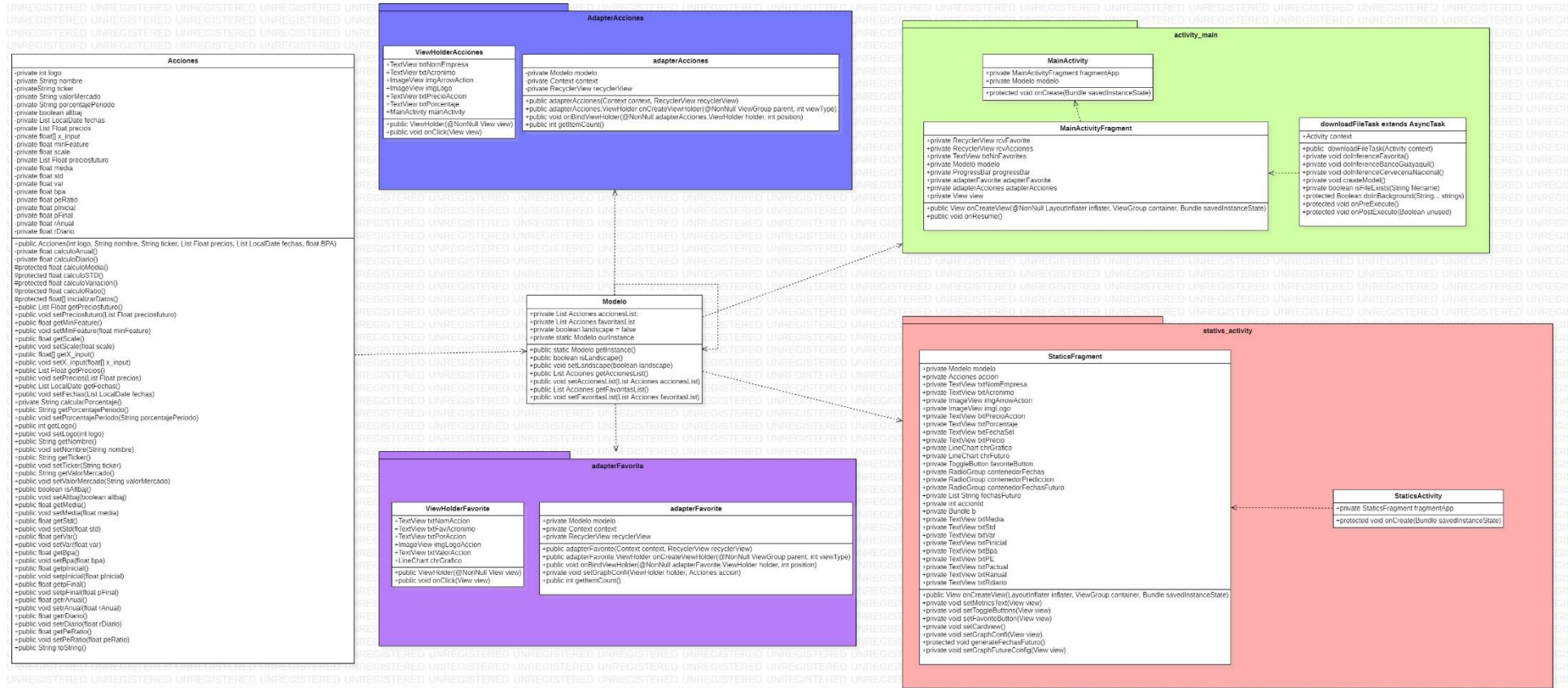


Figura #14: Diagrama UML de aplicación

Anexo B: Manual de Usuario

Ingreso a la aplicación: La primera vez que se ingresa a la aplicación, tras ser instalada, se requiere de una conexión a internet. Se espera a la descarga del archivo en donde el progreso se puede visualizar en el progress Bar presente. Una vez el progressBar desaparezca y se desplieguen los datos el usuario puede utilizar la aplicación.



Figura #15: Pantalla de inicio antes y después de carga de datos.

Selección de acción: El usuario puede seleccionar la acción de su interés dando clic en la misma tanto en la sección de favoritos como en la sección de mercado. En cada elemento se presenta el nombre de la acción, su ticker de cotización en bolsa, el rango de crecimiento con respecto al precio del cierre del mes pasado, y el precio actual de la acción.

Visualización datos y manipulación gráfico: En la pantalla Estadísticas se despliega toda la información sobre una acción. En la parte superior se presentan el nombre de la acción, su ticker de cotización en bolsa, el rango de crecimiento con respecto al precio del cierre del mes pasado, y el precio actual de la acción. Siguiendo a la parte media, se presentan dos botones. En los primeros podemos encontrar "Pasado" y "Futuro" el cual dando clic podemos cambiar la visualización de los datos entre históricos y predichos. Los siguientes botones son la selección del tiempo. Dando clic en el tiempo deseado se puede visualizar que le gráfico cambia con respecto al rango de tiempo seleccionado. En el caso de que no

cambie, significa que el gráfico está en movimiento. Se debe volver a seleccionar el rango de fecha deseado para que este haga efecto. Dentro del gráfico, el usuario puede mover a lo largo de las x, y además seleccionar un punto en el gráfico para poder visualizar la fecha y el precio. Finalmente, en la parte inferior podemos visualizar los datos estadísticos de cada acción. Estos no tienen interacción con el usuario.



Figura #16: Estructura de la pantalla Estadísticas

Selección acción favorita: para seleccionar y almacenar una acción como favorita, se debe dar clic en la estrella que se encuentra presente en la parte superior derecha, alado del título "Estadísticas". Una vez seleccionada esta será rellenada y en la pantalla principal se podrá visualizar la acción escogida.



Figura #17: Flujo de pantallas para agregar una acción como favorita

Anexo C: Uso de Modelo Keras en Android

Una vez entrenado el modelo se procede a realizar la exportación del modelo para ser utilizada dentro de nuestra aplicación Android Studio.

Para esto se va a utilizar la librería TensorflowLite. Es una librería que nos permite ejecutar modelos Keras o tensorflow en nuestra diapositiva android.

Lo primero que debemos hacer es exportar el modelo. Para esto se realiza una conversión de nuestro modelo Keras a tensorflowLite. Este converter de igual manera es brindada por la propia librería.

Una vez realizado la conversión del modelo, debemos comprobar que el modelo este obteniendo los mismos resultados que el modelo de Keras, por lo que se realiza una predicción con el Modelo Keras TensorFlow y el Modelo TensorFlow Lite y se valida que todos los resultados sean iguales.

```

Output exceeds the size limit. Open the full output data in a text editor
1/1 [=====] - 0s 24ms/step
Done. The result of TensorFlow matches the result of TensorFlow Lite.
1/1 [=====] - 0s 18ms/step
Done. The result of TensorFlow matches the result of TensorFlow Lite.
1/1 [=====] - 0s 21ms/step
Done. The result of TensorFlow matches the result of TensorFlow Lite.
1/1 [=====] - 0s 21ms/step
Done. The result of TensorFlow matches the result of TensorFlow Lite.
1/1 [=====] - 0s 21ms/step
Done. The result of TensorFlow matches the result of TensorFlow Lite.
1/1 [=====] - 0s 24ms/step
Done. The result of TensorFlow matches the result of TensorFlow Lite.
1/1 [=====] - 0s 20ms/step
Done. The result of TensorFlow matches the result of TensorFlow Lite.
1/1 [=====] - 0s 19ms/step
Done. The result of TensorFlow matches the result of TensorFlow Lite.
1/1 [=====] - 0s 20ms/step
Done. The result of TensorFlow matches the result of TensorFlow Lite.
1/1 [=====] - 0s 19ms/step
Done. The result of TensorFlow matches the result of TensorFlow Lite.
1/1 [=====] - 0s 20ms/step
Done. The result of TensorFlow matches the result of TensorFlow Lite.
1/1 [=====] - 0s 23ms/step
Done. The result of TensorFlow matches the result of TensorFlow Lite.
1/1 [=====] - 0s 24ms/step
...
1/1 [=====] - 0s 23ms/step
Done. The result of TensorFlow matches the result of TensorFlow Lite.
1/1 [=====] - 0s 23ms/step
Done. The result of TensorFlow matches the result of TensorFlow Lite.

```

Figura #18: Comparación resultados Tensorflow Lite con modelo Keras

Una vez Exportado el modelo, la librería mismo nos dice que código utilizar y que librerías exportar para poder utilizar el modelo. Debemos tener en cuenta como es el input del modelo y el output del modelo.

```

ModelCF model =ModelCF.newInstance(context);
TensorBuffer inputFeature0=TensorBuffer.createFixedSize(new int[]{1,60,1}, DataType.FLOAT32);
inputFeature0.loadArray(modelo.getAccionesList().get(2).getX_input());
ModelCF.Outputs outputs=model.process(inputFeature0);
TensorBuffer outputFeature0=outputs.getOutputFeature0AsTensorBuffer();
data_predicted=outputFeature0.getFloatArray();
model.close();

```

Figura #19: Código que demuestra el uso del modelo “tsflite” en Android Studio

Primero debemos realizar con el mismo rango de entrenamiento, la normalización min max de los datos, y se almacena el escalador que es el nuevo max menos el nuevo mínimo sobre el actual max y mínimo y el minFeature el cual se obtiene calculando el nuevo mínimo menos el mínimo actual por el escalador. Estos valores nos van a ayudar luego para desnormalizar los datos.

En nuestro caso, el input que pide el modelo es una Matriz de tres dimensiones del tipo (1, 60, 1) Definimos esto en un Float simple propio de Java ya que tensorflow no admite ArrayList, solo arrays nativos, se ingresan los 60 últimos precios obtenidos por cada acción y se realiza la predicción. Estos datos se agregan a al modelo mediante la función del tensorbuffer loadarray. Ya que nuestro tensorbuffer tiene un shape definido, la función loadarray se encarga de redimensionar nuestro arreglo a 3 dimensiones para que este pueda ser interpretado por el modelo.

De igual manera debemos tener en cuenta el shape de nuestro output, en nuestro caso simplemente es un arreglo con 30 valores.

Una vez realizada la predicción, desnormalizamos los datos restando a cada valor el min feature y dividiéndolo para el escalador. Finalmente, los datos son almacenados en una lista que contiene cada acción para luego utilizar estos datos en el linechart.

Obtenemos los siguientes resultados del modelo de la red, y comparando con los resultados obtenidos en Python realizando la predicción con los mismos datos de Android, podemos observar que el modelo en Android está realizando las predicciones de manera correcta y eficiente.

Anexo D: Código Fuente

El código fuente de la aplicación se encuentra dividido en dos segmentos. El repositorio que consta con el código del proyecto móvil Android studio es el siguiente enlace:

<https://github.com/jedavila/NeuronalStockApp> y el repositorio que contiene el entrenamiento de la red neuronal es el siguiente <https://github.com/jedavila/NeuronalStockNotebook>.

Dentro del repositorio de Android studio nos encontraremos con los archivos correspondientes a las clases de java para la programación de la aplicación, además de los layouts utilizados en la misma.

Finalmente, para el repositorio de python, contiene un archivo “.ipynb” el cual es un notebook de python en donde se presenta el flujo de desarrollo para la red neuronal.