Práctico 1

El objetivo de este práctico será buscar la red neuronal que mejor prediga el tiempo de adopción de las mascotas. El tiempo de adopción puede adoptar cinco valores: (0,1,2,3,4).

1)

Modelo Base

Para empezar, generamos una red neuronal con una sola capa oculta compuesta por 256 neuronas. El input son solo las columnas Age, Fee y Gender. Las primeras dos son variables numéricas, en cambio la última es categórica y asume tres valores distintos, por lo tanto se optó por aplicarle one hot encoding a esta, resultando el input de la red neuronal en 5 columnas. En tanto, a las columnas numéricas se les aplico el método minmax para escalarlas. Para la activación de la capa oculta se eligió la función ReLu en cambio que para la final se eligió una función lineal. El número de epochs es 100, el batch size 32. En cuanto al optimizador, se eligió Adam, que es el mismo que se usará para todos los modelos, ya que combina propiedades deseables de otros optimizadores y funciona bien para la mayoría de los problemas.

```
model = Sequential()
n_cols = 5
model.add(Dense(hidden_n,activation="relu",input_shape=(n_cols,)))
model.add(Dense(5,activation="linear"))
model.compile(optimizer="adam", loss='mean_squared_error',metrics=["acc"])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 256)               1536
_________________________________________________________________
dense_7 (Dense)              (None, 5)                 1285
=================================================================
Total params: 2,821
Trainable params: 2,821
Non-trainable params: 0
_________________________________________________________________
```

Evaluando sobre el conjunto de dev:
Accuracy: 0.282
Loss: 0.166

Modelo 2

Para mejorar el modelo anterior, decidimos cambiar la función de activación de la última capa por una softmax, que asigna probabilidades de que el output pertenezca a una de las clases en las que se quiere clasificar. En la práctica esta función presenta mejoras con respecto a la linear. A su vez, decidimos cambiar la función de pérdida por cross entropy, ya que esta se ajusta mejor a modelos en los que el output es una probabilidad.

Al cambiar solo la función de activación y la de pérdida, el número de parámetros a entrenar será el mismo que en el modelo base.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 256)               1536
_________________________________________________________________
dense_9 (Dense)              (None, 5)                 1285
=================================================================
Total params: 2,821
Trainable params: 2,821
Non-trainable params: 0
_________________________________________________________________
```

Evaluando sobre el conjunto de dev:
Accuracy: 0.248
Loss: 1.645

Modelo 3

Para mejorar la performance del modelo 2, decidimos agregar mas capas ocultas y dropout de 0.3 (para evitar que sobreajuste a los datos de entrenamiento). Se agrega una capa para hacer batch normalization entre las distintas capas, esto ayuda a reducir el overfitting y permite usar un dropout no tan alto y de esta forma no perder tanta información. Por último, se introduce un parámetro de regularización (0.01) en las capas ocultas, para reducir el overfitting.

```
model = Sequential()
model.add(Dense(hidden_n,activation="relu",input_shape=(n_cols,),kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(Dropout(args.dropout[0]))
model.add(Dense(hidden_n/
2,activation="relu",input_shape=(n_cols,),kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(Dropout(args.dropout[0]))
model.add(Dense(5,activation="softmax"))
model.compile(optimizer="adam", loss='categorical_crossentropy',metrics=["acc"])
```

Layer (type)	Output Shape	Param #
dense_17 (Dense)	(None, 256)	1536
batch_normalization (BatchNo	(None, 256)	1024
dropout_4 (Dropout)	(None, 256)	0
dense_18 (Dense)	(None, 128)	32896
batch_normalization_1 (Batch	(None, 128)	512
dropout_5 (Dropout)	(None, 128)	0
dense_19 (Dense)	(None, 5)	645

Total params: 36,613
Trainable params: 35,845
Non-trainable params: 768

Evaluando sobre el conjunto de dev:
Accuracy: 0.339
Loss: 1.446

Como se puede observar, el modelo 3 presentó una gran mejora en el accuracy sobre el conjunto dev, con respecto a los modelos 1 y 2.

2)

Modelo 4

A continuación se evaluará como afectan a los resultados del modelo el agregar o modificar las columnas del dataset. Para empezar, agregaremos las columnas MaturitySize, Health y Quantity, todas con valores numéricos. Si bien Health asume solo 3 valores, vamos a aplicarle el método de escalado minmax igual que al resto, por suponer que los valores que asume son ordinales.

Layer (type)	Output Shape	Param #
dense_23 (Dense)	(None, 256)	2304
batch_normalization_4 (Batch	(None, 256)	1024
dropout_8 (Dropout)	(None, 256)	0
dense_24 (Dense)	(None, 128)	32896
batch_normalization_5 (Batch	(None, 128)	512
dropout_9 (Dropout)	(None, 128)	0
dense_25 (Dense)	(None, 5)	645

Total params: 37,381
Trainable params: 36,613
Non-trainable params: 768

Accuracy: 0.282
Loss: 1.499

Modelo 5

Los resultados del modelo 4 no parecen mejorar con respecto al modelo 3 (probablemente la asunción de que health era ordinal no sea correcta). Por lo tanto pasaremos a tomar en cuenta la columna Breed1. Esta columna es categórica y contiene datos de la raza de la mascota en cuestión, pero el problema es que tiene demasiadas categorías, por lo que hacer un one hot enconding sobre esta llevaría a agregar muchas columnas con valores dispersos. Para agregar esta columna a la red neuronal de una forma conveniente, se le aplicará un embedding que permite reducir la dimensionalidad de los datos.
La columna Breed1 se pasará primero por una capa de embedding y luego se unirá a la parte de la red que analiza los datos numéricos.

```
__________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
==========================================================================================
input_26 (InputLayer)        [(None, 1)]               0
__________________________________________________________________________________________
embedding_17 (Embedding)     (None, 1, 76)             23408       input_26[0][0]
__________________________________________________________________________________________
input_27 (InputLayer)        [(None, 8)]               0
__________________________________________________________________________________________
flatten_11 (Flatten)         (None, 76)                0           embedding_17[0][0]
__________________________________________________________________________________________
dense_59 (Dense)             (None, 256)               2304        input_27[0][0]
__________________________________________________________________________________________
concatenate_21 (Concatenate) (None, 332)               0           flatten_11[0][0]
                                                                   dense_59[0][0]
__________________________________________________________________________________________
dense_60 (Dense)             (None, 128)               42624       concatenate_21[0][0]
__________________________________________________________________________________________
batch_normalization_20 (BatchNo (None, 128)            512         dense_60[0][0]
__________________________________________________________________________________________
dropout_17 (Dropout)         (None, 128)               0           batch_normalization_20[0][0]
__________________________________________________________________________________________
dense_61 (Dense)             (None, 5)                 645         dropout_17[0][0]
==========================================================================================
Total params: 69,493
Trainable params: 69,237
Non-trainable params: 256
__________________________________________________________________________________________
```

Accuracy: 0.339
Loss: 2.0730

La accuracy aumenta considerablemente respecto al modelo anterior, y es similar a la del modelo 3, por lo que podemos suponer que la raza de las mascotas es un factor importante al momento de la elección, al igual que la edad, el sexo y el pago que haya que realizar. Ya que este modelo tiene mas

columnas que el modelo 3 e igual accuracy, por parsimonia decidimos quedarnos con el modelo 3 y utilizar este para la competencia de Kaggle, lo que dio un score de 0.33.