-NN para detectar Fake news

-Utilizamos dos datasets que contienen el texto de noticias a los que les creamos una clase para las reales (0) y las falsas (1)

-Visualizamos las palabras más frecuentes en cada clase

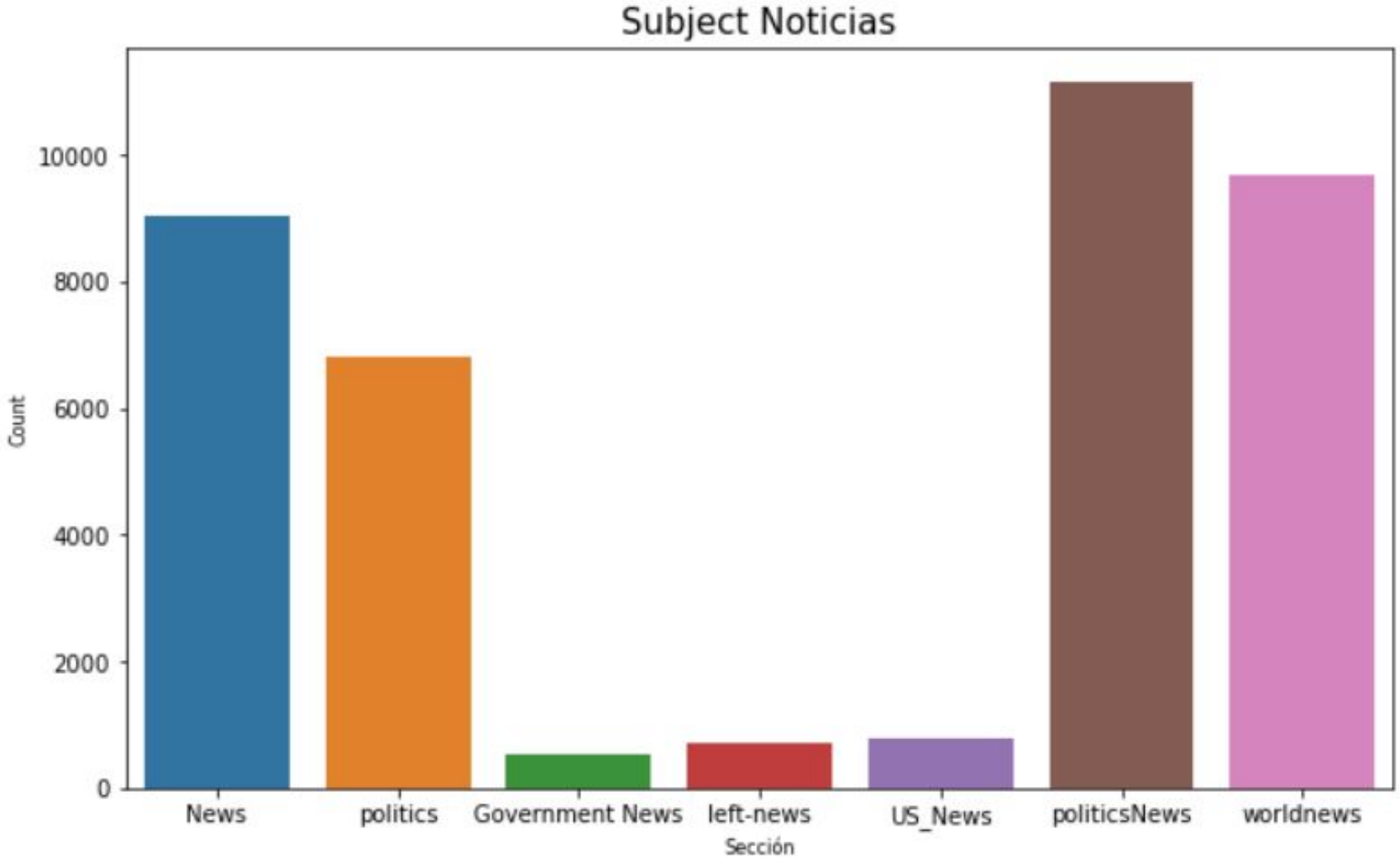
FAKE

TRUE

-En la nube de palabras de noticias verdaderas aparecen palabras que indican condicional.
Podría interpretarse como que en noticias falsas no es relevante la rigurosidad de que algo esté confirmado o no.

-Aparecen de forma más notoria palabras como WATCH y VIDEO en la nube de fake news

-En las noticias verdaderas hay palabras como OFFICIAL o SOURCE y en las fake, no


Subject Noticias
10000
8000
6000
4000
2000
0
Count
News
politics
Government News
left-news
US_News
politicsNews
worldnews
Sección

-Detectamos duplicados, unimos título y texto de la noticia en una misma variable

-A los subjects que figuran con diferente nombre en cada dataset los unimos a una misma categoría para que no queden subjects exclusivos de cada clase

[27]:
```
#unificamos categorías similares

subject_replace = {
    "politicsNews" : "politics",
    "US_News" : "News",
    "Government News" : "politics",
    "left-news" : "politics"
     }

df["new_subject"] = df["subject"]

for key in subject_replace:
    df["new_subject"] = df.new_subject.str.replace(key, subject_replace[key])

df.new_subject.value_counts(1)
```

[27]:
```
politics     0.493338
worldnews    0.255210
News         0.251451
Name: new_subject, dtype: float64
```

-Generamos más variables que extraemos de la fecha

new_subject_News	new_subject_politics	new_subject_worldnews	year	news	month_Apr	month_Aug	month_Dec	month_Feb	month_Jan	month_Jul
1	0	0	2017	Donald Trump Sends Out Embarrassing New Year'...	0	0	1	0	0	0
1	0	0	2017	Drunk Bragging Trump Staffer Started Russian ...	0	0	1	0	0	0
				Sheriff David Clarke						

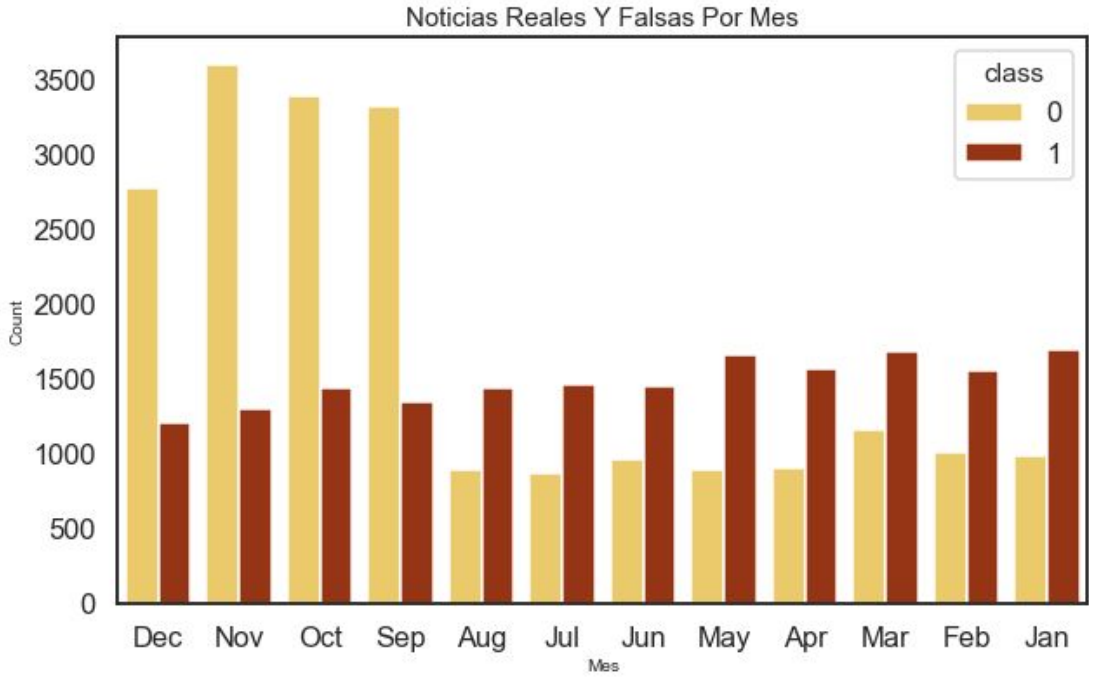
Nos fijamos la cantidad de datos que hay para cada mes y año

```
[31]: #Extraemos los anios
pattern_year=r'(\d{4})'
df['year']=df.date.str.extract(pattern_year)
df.year.value_counts()
```

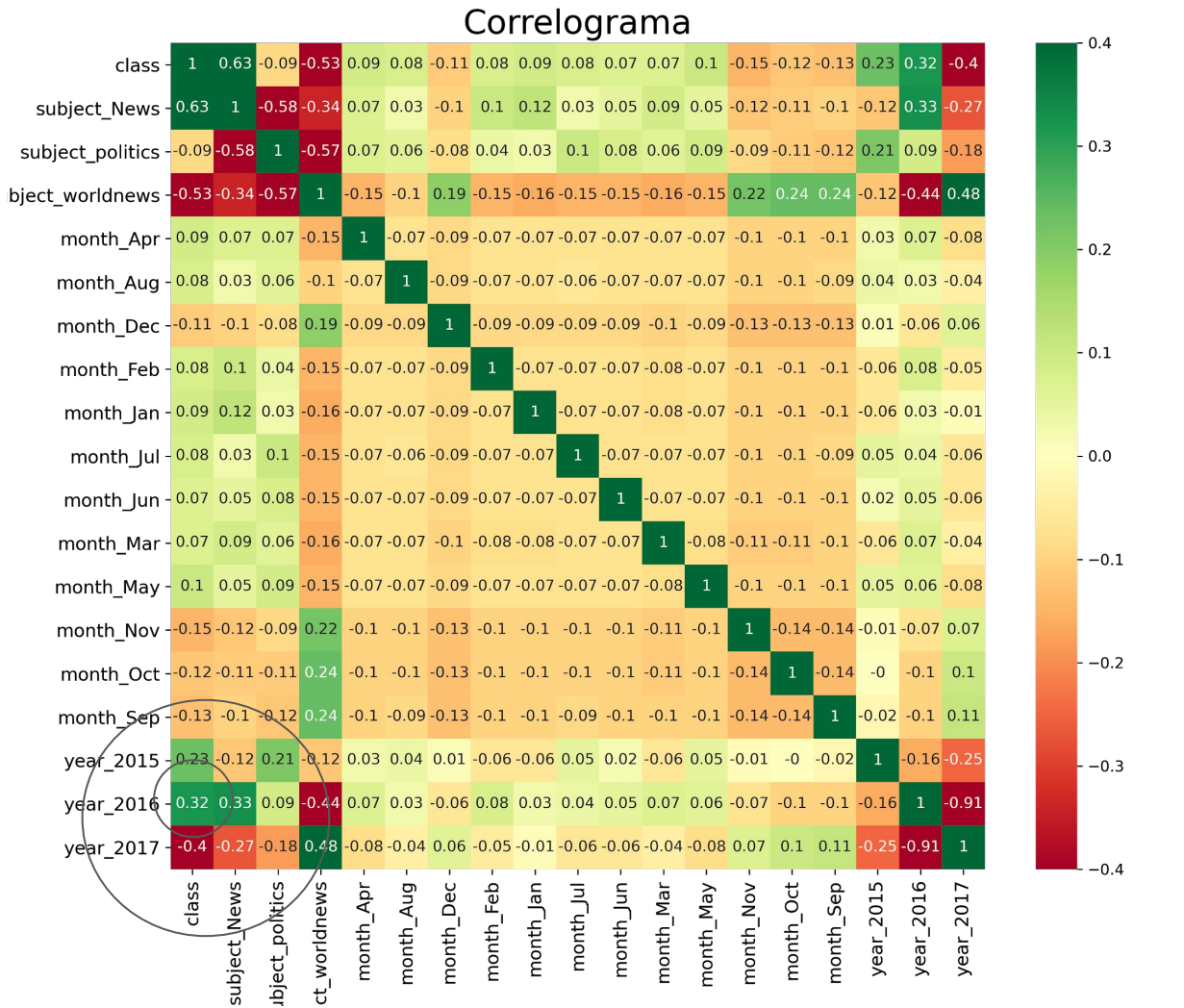
```
[31]: 2017    22951
2016    14093
2015     1646
Name: year, dtype: int64
```

2017 es el año con más noticias pero no el año con más fake news

Mayo, Marzo y Enero son los meses con más fake news. Las elecciones son en Noviembre



2016 es el año con más fake news. Tiene sentido porque fue el año de las elecciones



Bag of words/CountVectorizer

-Iteramos sobre cada oración para pasar el texto de las noticias a minúscula, remover puntos y espacios

-Tokenizamos las oraciones y creamos un diccionario que contiene la frecuencia con la que aparecen las palabras

-Transformamos las oraciones a una representación vectorial para que aparezca 1 si esa palabra está en la oración y 0 si no está

-Probamos hacer esto de forma manual y finalmente usamos CountVectorizer porque tardaba menos. Agregamos stopwords

https://stackabuse.com/python-for-nlp-creating-bag-of-words-model-from-scratch/

Cosas que fuimos probando:

-Pasarle a la Red Neuronal sólo las palabras vectorizadas para que clasifique

-Pasarle las palabras vectorizadas + el resto de las variables (meses, subject) y comparar si había diferencia

-Red neuronal con regularización L1, L2, dropout rate (sin regularizar overfitteaba)

-La misma red neuronal con 1 y 2 capas. Nos quedamos con la de una capa

-Sumamos KerasClassifier, EarlyStopping, ReduceLROnPlateau, RandomizedSearch y cross validamos con 3 folds

Modelo sin callbacks ni regularización

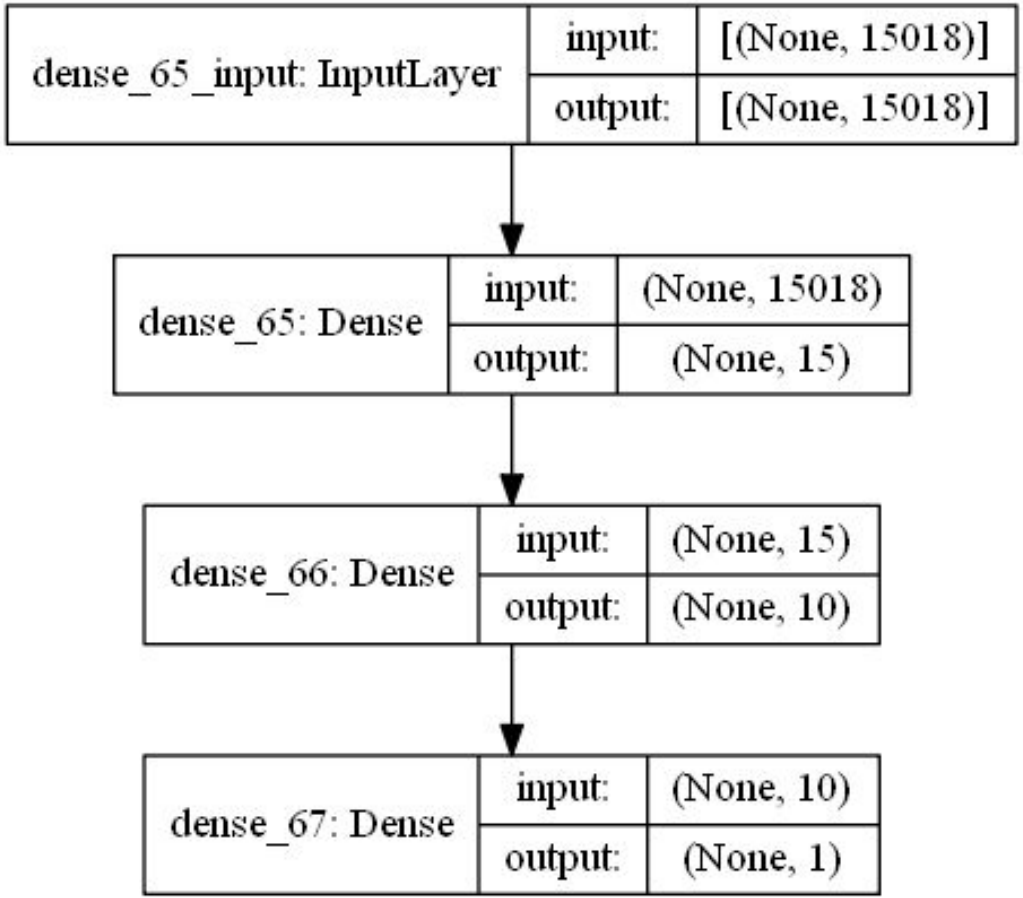
[59]:
```
#Compilamos el modelo
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

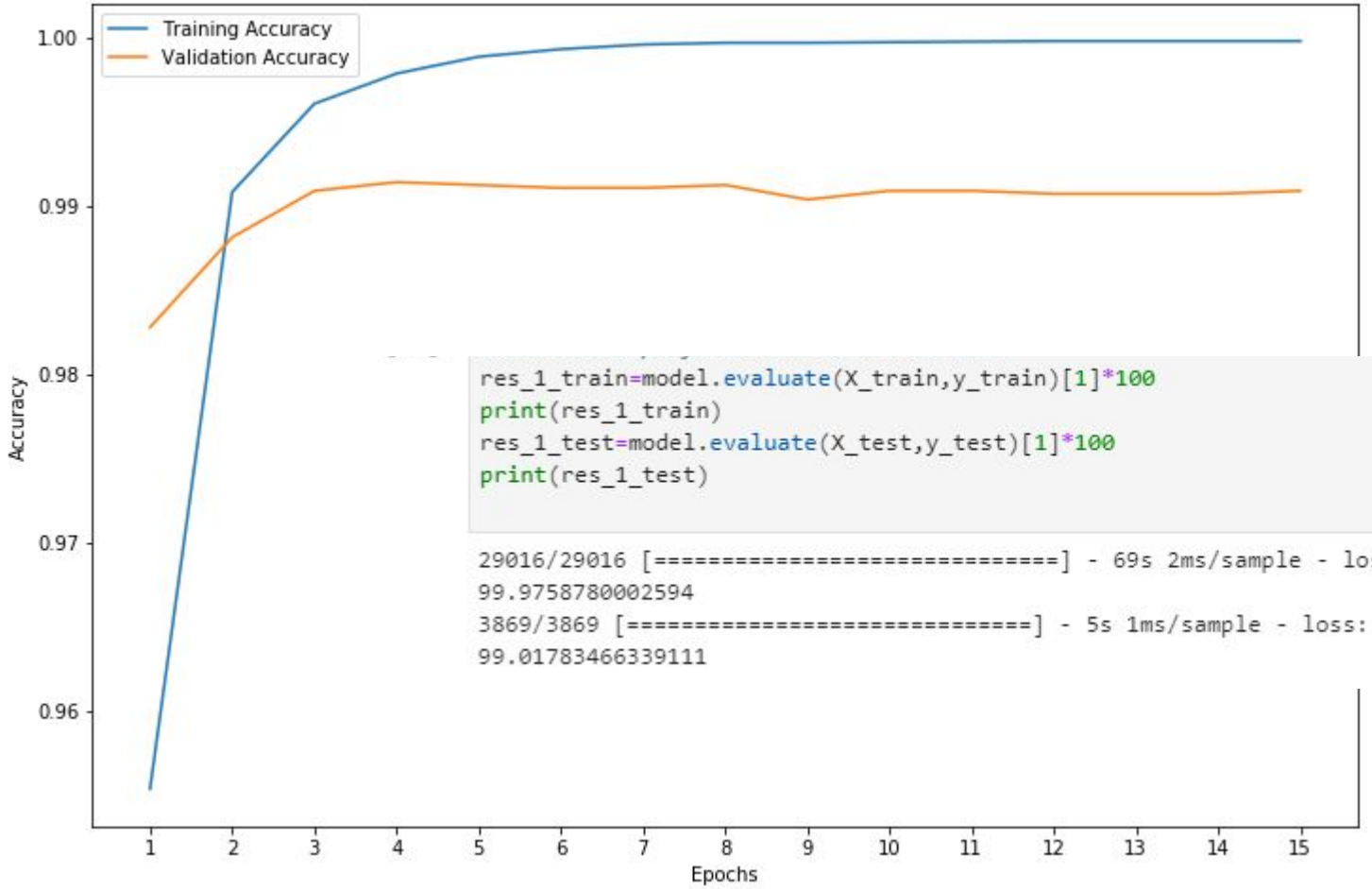
```
WARNING:tensorflow:From C:\Anaconda\envs\dhdsblend\lib\site-packages\tensorflow\python\ops\nn_impl.py:180: add_dispatch_support.<
er (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

[60]:
```
#Entrenamos el modelo sin callbacks
history=model.fit(x=X_train,y=y_train,
                  epochs=15,batch_size=512,validation_data=(X_val,y_val))
```

```
Train on 29016 samples, validate on 5803 samples
Epoch 1/15
29016/29016 [==============================] - 55s 2ms/sample - loss: 0.3156 - acc: 0.9554 - val_loss: 0.2087 - val_acc: 0.9828
Epoch 2/15
29016/29016 [==============================] - 45s 2ms/sample - loss: 0.1583 - acc: 0.9908 - val_loss: 0.1333 - val_acc: 0.9881
Epoch 3/15
29016/29016 [==============================] - 44s 2ms/sample - loss: 0.0995 - acc: 0.9960 - val_loss: 0.0939 - val_acc: 0.9909
Epoch 4/15
29016/29016 [==============================] - 45s 2ms/sample - loss: 0.0687 - acc: 0.9978 - val_loss: 0.0723 - val_acc: 0.9914
Epoch 5/15
29016/29016 [==============================] - 42s 1ms/sample - loss: 0.0496 - acc: 0.9988 - val_loss: 0.0589 - val_acc: 0.9912
Epoch 6/15
29016/29016 [==============================] - 44s 1ms/sample - loss: 0.0368 - acc: 0.9993 - val_loss: 0.0530 - val_acc: 0.9910
Epoch 7/15
29016/29016 [==============================] - 43s 1ms/sample - loss: 0.0277 - acc: 0.9996 - val_loss: 0.0472 - val_acc: 0.9910
Epoch 8/15
29016/29016 [==============================] - 43s 1ms/sample - loss: 0.0211 - acc: 0.9997 - val_loss: 0.0445 - val_acc: 0.9912
Epoch 9/15
```

Red Neuronal sin callbacks ni regularización





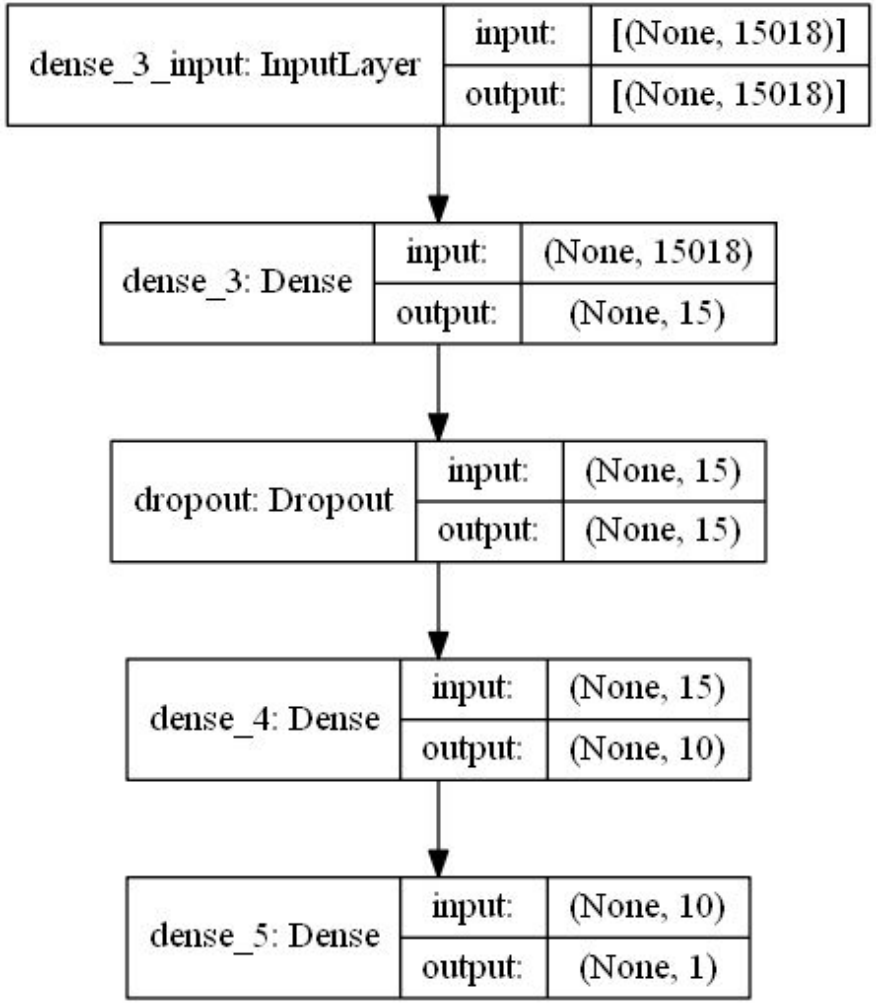

```
res_1_train=model.evaluate(X_train,y_train)[1]*100
print(res_1_train)
res_1_test=model.evaluate(X_test,y_test)[1]*100
print(res_1_test)
```

```
29016/29016 [==============================] - 69s 2ms/sample - loss: 0.0040 - acc: 0.9998
99.9758780002594
3869/3869 [==============================] - 5s 1ms/sample - loss: 0.0440 - acc: 0.9902
99.01783466339111
```

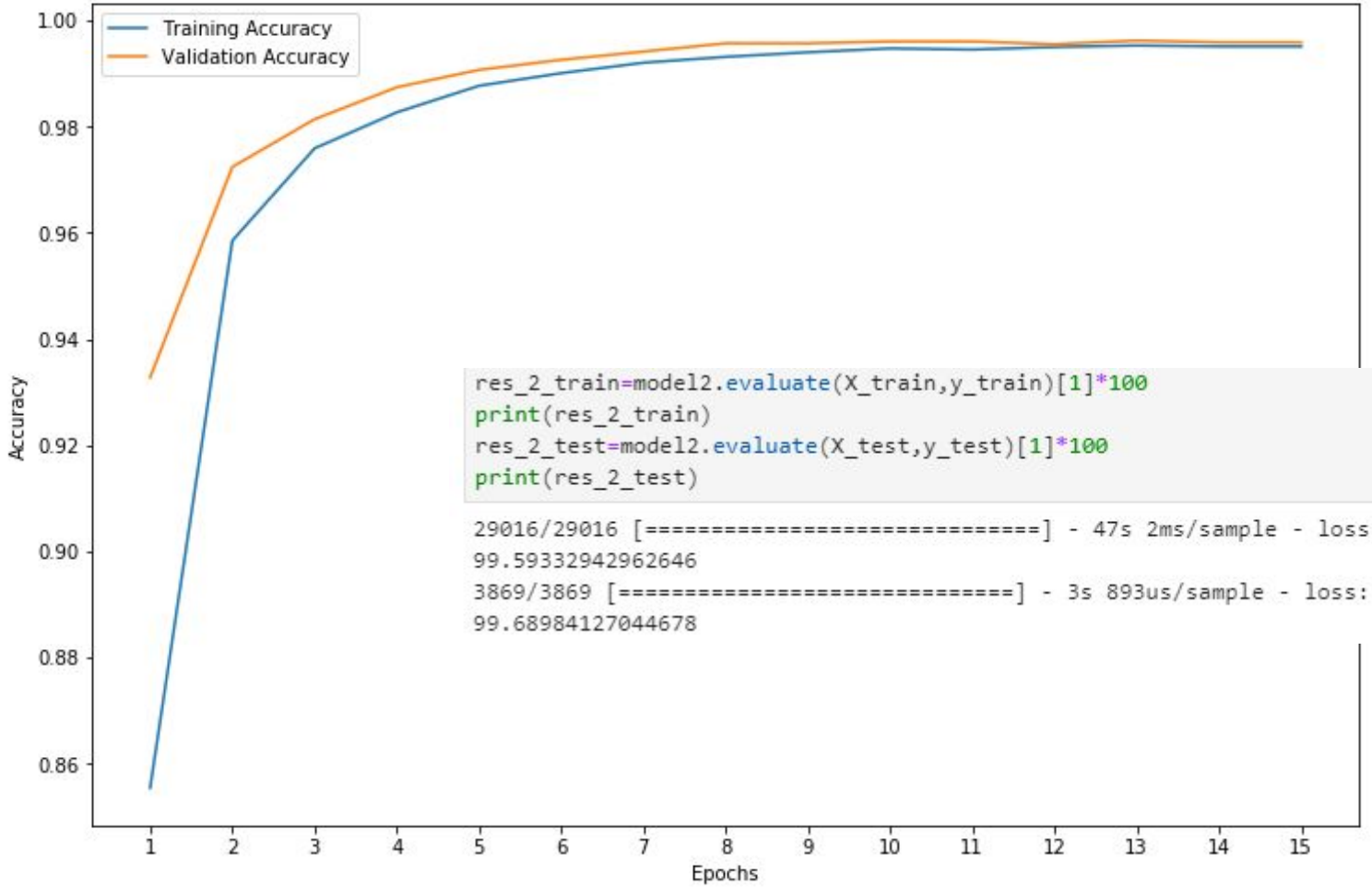
Modelo con regularización y Dropout rate

[77]:
```
history2=model2.fit(X_train,
                    y_train,
                    epochs=15,
                    batch_size=512,
                    validation_data=(X_val,y_val))
```

```
Train on 29016 samples, validate on 5803 samples
Epoch 1/15
29016/29016 [==============================] - 63s 2ms/sample - loss: 0.7833 - acc: 0.8555 - val_loss: 0.5343 - val_acc: 0.9328
Epoch 2/15
29016/29016 [==============================] - 44s 2ms/sample - loss: 0.4819 - acc: 0.9585 - val_loss: 0.4263 - val_acc: 0.9724
Epoch 3/15
29016/29016 [==============================] - 45s 2ms/sample - loss: 0.4000 - acc: 0.9759 - val_loss: 0.3648 - val_acc: 0.9814
Epoch 4/15
29016/29016 [==============================] - 45s 2ms/sample - loss: 0.3475 - acc: 0.9827 - val_loss: 0.3266 - val_acc: 0.9874
Epoch 5/15
29016/29016 [==============================] - 47s 2ms/sample - loss: 0.3123 - acc: 0.9877 - val_loss: 0.2915 - val_acc: 0.9907
Epoch 6/15
29016/29016 [==============================] - 47s 2ms/sample - loss: 0.2879 - acc: 0.9901 - val_loss: 0.2671 - val_acc: 0.9926
Epoch 7/15
29016/29016 [==============================] - 45s 2ms/sample - loss: 0.2681 - acc: 0.9920 - val_loss: 0.2505 - val_acc: 0.9941
Epoch 8/15
29016/29016 [==============================] - 45s 2ms/sample - loss: 0.2537 - acc: 0.9931 - val_loss: 0.2379 - val_acc: 0.9957
Epoch 9/15
29016/29016 [==============================] - 44s 2ms/sample - loss: 0.2417 - acc: 0.9940 - val_loss: 0.2299 - val_acc: 0.9957
Epoch 10/15
29016/29016 [==============================] - 45s 2ms/sample - loss: 0.2341 - acc: 0.9947 - val_loss: 0.2293 - val_acc: 0.9960
Epoch 11/15
29016/29016 [==============================] - 44s 2ms/sample - loss: 0.2274 - acc: 0.9945 - val_loss: 0.2195 - val_acc: 0.9960
```



Red Neuronal con regularización y Dropout




```
res_2_train=model2.evaluate(X_train,y_train)[1]*100
print(res_2_train)
res_2_test=model2.evaluate(X_test,y_test)[1]*100
print(res_2_test)
```

```
29016/29016 [==============================] - 47s 2ms/sample - loss: 0.2034 - acc: 0.9959
99.59332942962646
3869/3869 [==============================] - 3s 893us/sample - loss: 0.2014 - acc: 0.9969
99.68984127044678
```

Con Keras Classifier, Early Stopping, ReduceLROnPlateau, Cross validation y probando diferentes dropout rates y units

[84]:
```
y_pred_grid =grid.predict(X_test)
res_3_test=accuracy_score(y_test, y_pred_grid)
print('Accuracy=', res_3_test)

y_train_grid=grid.predict(X_train)
res_3_train=accuracy_score(y_train, y_train_grid)
print('Accuracy=', res_3_train)
```

```
3869/3869 [==============================] - 3s 658us/sample
Accuracy= 0.9968984233652106
29016/29016 [==============================] - 37s 1ms/sample
Accuracy= 0.9957264957264957
```

[85]:
```
df_Results.loc[3,'Model']='Modelo 3'
df_Results.loc[3,'Train Acc']=res_3_train
df_Results.loc[3,'Test Acc']=res_3_test
df_Results.loc[3,'Train Acc - Test Acc']=res_3_train-res_3_test

df_Results
```

[85]:

	Model	Train Acc	Test Acc	Train Acc - Test Acc
1	Modelo 1	99.969	99.1212	0.847763
2	Modelo 2	99.5726	99.6123	-0.039655
3	Modelo 3	0.995726	0.996898	-0.00117193

```
print(grid.best_score_.round(5))
print(grid.best_estimator_.get_params())
```

```
0.99552
{'batch_size': 512, 'epochs': 15, 'verbose': 1, 'units': 10, 'dropout_rate': 0.5,
```