1) Feed Foward

Para a rede feed foward foi usado:

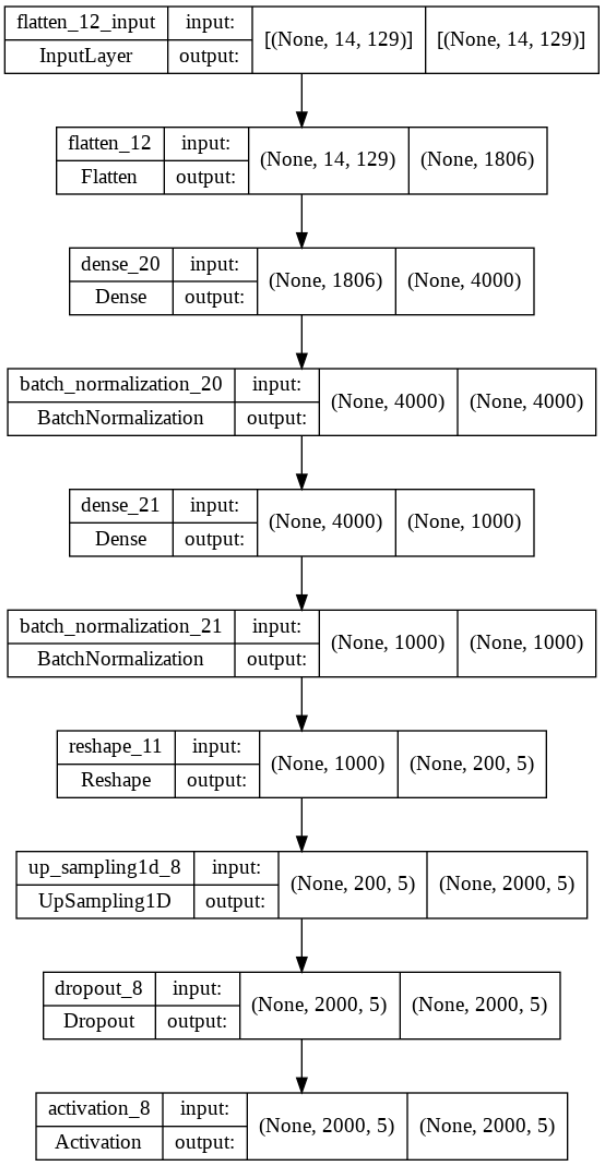
- a função randomExtract(x) para extrair um sinal aleatório de tamanho a definir (neste caso usei L=2000);

- Short-time Fourier Transform com frame_length 256 e frame_step 128;

- um batch size de 250, para acelerar o treino e aumentar a convergência, 10 steps por epoca;

- Adam optimizer com parâmetros standard revelou-se melhor do que todos os SGD que experimentei devido ao seu learning rate adaptativo. Optimizers SGD experimentados tinham cerca de 0.001 learning rate, 0.9 momentum, às vezes com decay outra vezes sem decay (se usasse decay ajustava o learning rate inicial para 0.1/0.01 dependendo do quão grande fosse o decay).

O modelo usado foi:



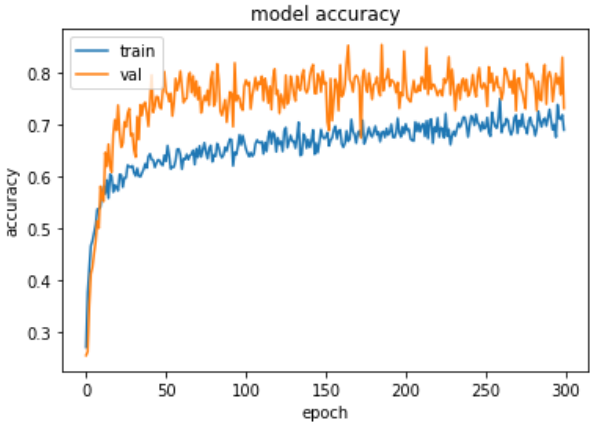

```
Total params: 11,249,000
Trainable params: 11,239,000
Non-trainable params: 10,000
```

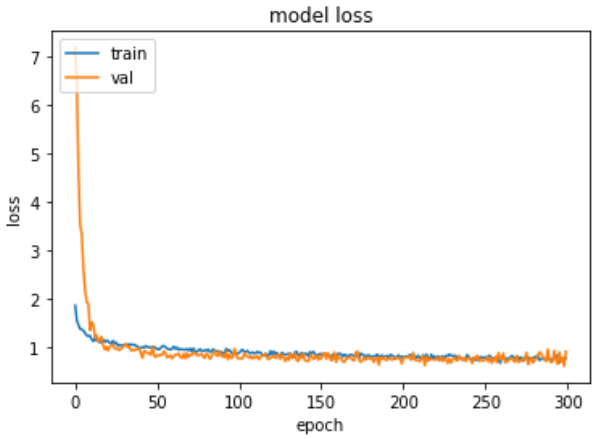
Começando com flatten para conseguir posteriormente aplicar layers Dense. Através de vários testes o modelo parecia comportar-se melhor com duas camadas de layers Dense. De seguida o número de camadas de cada Dense também foi ajustado através de testes, parecendo o melhor comportamento para 4000 camadas seguido de 1000 camadas.

Para reconstituir o sinal de 2000 de comprimento dividido em 5 classes foi feito um reshape depois da última dense. Dividindo as 1000 camadas da última layer dense por 5 obtemos um reshape de (200,5). Para chegar ao shape desejado basta aplicar um Upsampling de 10, desta forma (200x10,5) deu (2000,5).

Por último temos a ativação softmax por estarmos a resolver um problema de multiclass classification, de 5 classes (0, 1, 2, 3 e 4).

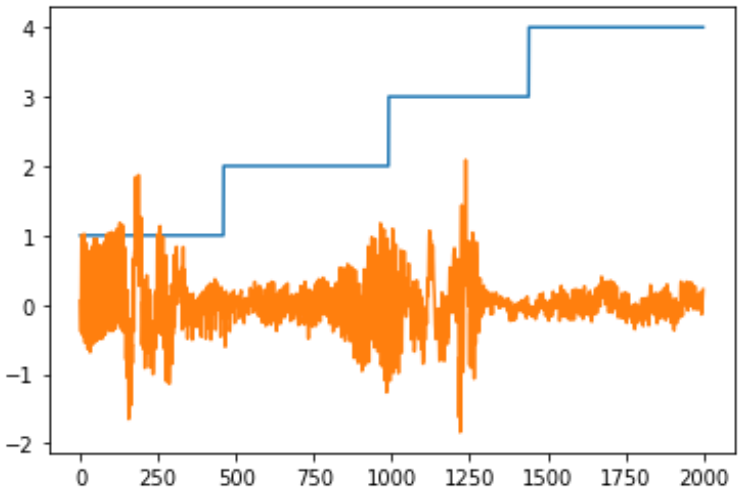
<u>Os resultados obtidos foram os seguintes:</u>






```
Train Accuracy
0.732515811920166
Validation Accuracy
0.8283723592758179
Train Loss
0.6809255480766296
Validation Loss
0.5776512026786804
```

Previsão:



2) Convolutional

Para a rede convolucional foi usado:

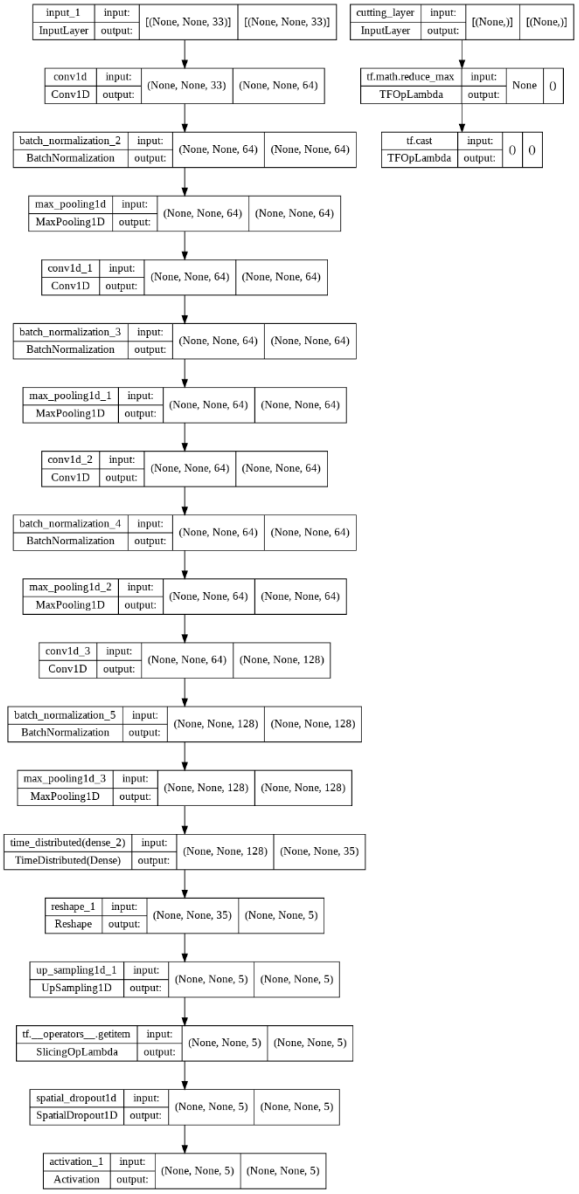
- Todos os ficheiros para treino, validação e teste, com tamanho variável;

- Short-time Fourier Transform com frame_length 64 e frame_step 32, torna o treino significativamente mais lento mas teoricamente melhora generalização melhorando os resultados obtidos;

- um batch size de 50, com 50 passos por época (50 x 50=2500) para garantir também uma maior generalização

- Adam optimizer com parâmetros standard revelou-se melhor do que todos os SGD que experimentei. Optimizer SGD experimentado tinha cerca de 0.001 learning rate, 0.9 momentum, às vezes com decay outra vezes sem decay (se usasse decay ajustava o learning rate inicial para 0.1/0.01 dependendo do quão grande fosse o decay).

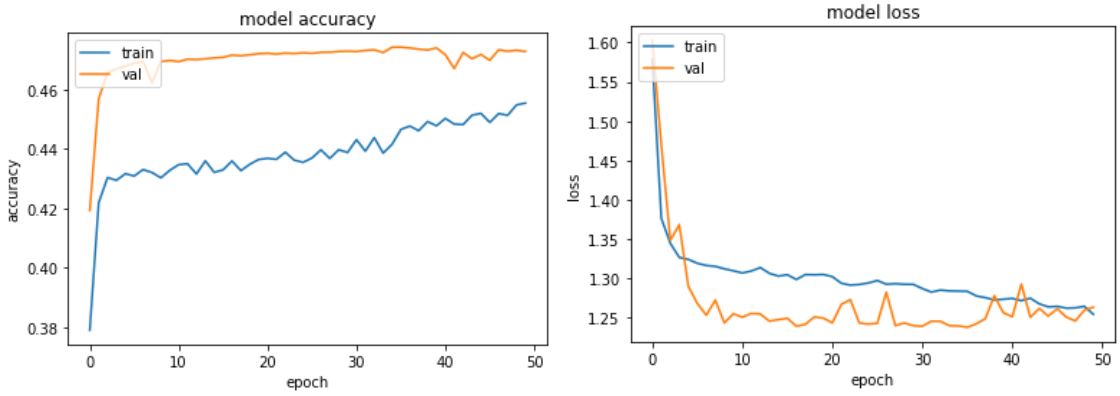
O modelo usado foi:




```
Total params: 61,603
Trainable params: 60,963
Non-trainable params: 640
```

Comecei com uma layer convolucional, seguida de batchnormalization, seguido de maxpooling. Repeti estas 3 layers por 3x, sempre que fiz um maxpooling com kernel 2 (reduzi a matriz para metade) dupliquei as camadas da layer convolucional. Depois de todas essas camadas tenho uma timedistributed com dense dentro, de 35 camadas, que é o múltiplo de 5 mais próximo do input do modelo (múltiplo de 5 porque estamos sempre a trabalhar com 5 classes de output). De seguida tenho o upsampling, que depende do tamanho que recebe do output da layer anterior, ou seja: se eu tiver uma matriz 26x5 repetida 12x (sendo 12 o tamanho do output da layer anterior) e quiser um output de comprimento 2000 tenho que fazer $\frac{2000}{26\times5\times12} = \frac{2000}{312} = 7$ ou seja Upsampling seria no mínimo 7 neste caso. Após o upsampling é necessário cortar o sinal pelo respetivo tamanho que é armazenado na layer do lado direito chamada "cutting_layer". Por último tenho ativação softmax tal como na rede feed foward.

<u>Os resultados obtidos foram os seguintes:</u>

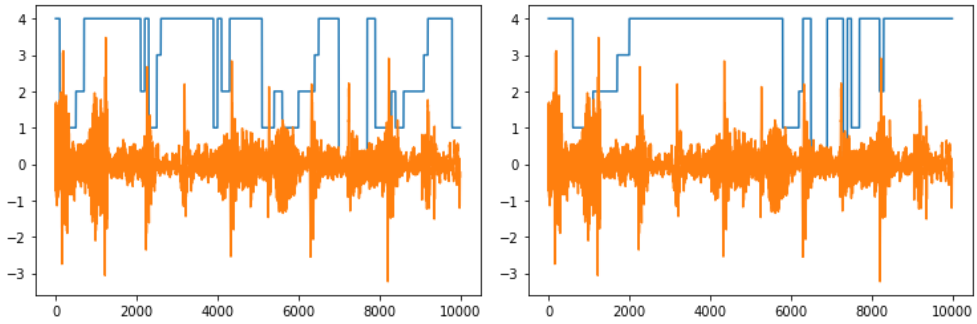



```
val accuracy:
0.4742903709411621
accuracy:
0.45546814799308777
val loss:
1.2379090785980225
train loss:
1.2544071674346924
```

Com valores de treino:

```
286/286 [==============================] - 176s 616ms/step - loss: 1.4594 - sparse_categorical_accuracy: 0.3968
Loss: 1.4593632221221924
Accuracy: 0.3968406915664673
```

Previsões:



3) Recurrent

Para a rede recorrente foi usado:

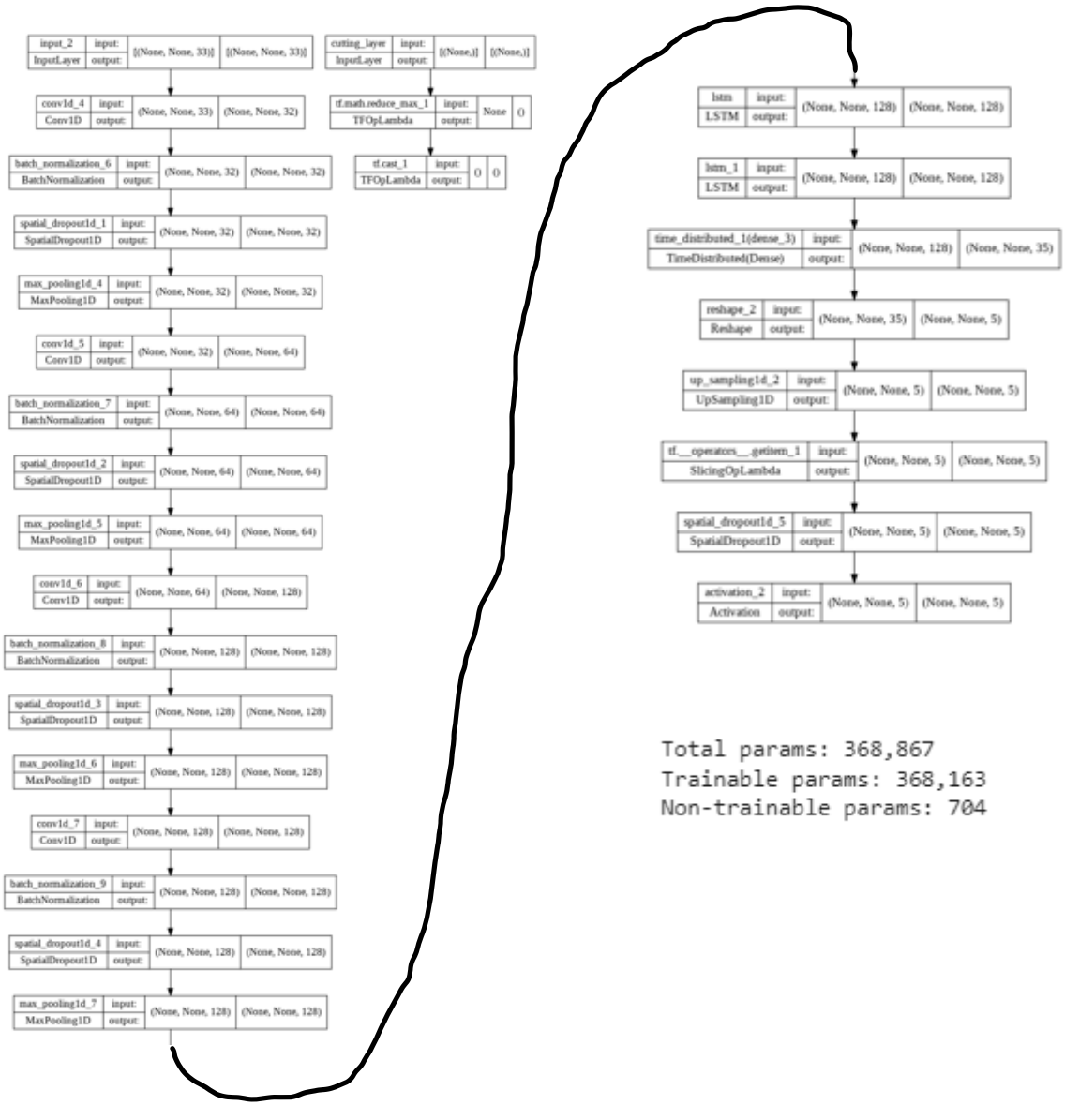
- Todos os ficheiros para treino, validação e teste, com tamanho variável;

- Short-time Fourier Transform com frame_length 64 e frame_step 32, torna o treino significativamente mais lento mas teoricamente melhora generalização melhorando os resultados obtidos;

- um batch size de 50, com 50 passos por época (50 x 50=2500) para garantir também uma maior generalização

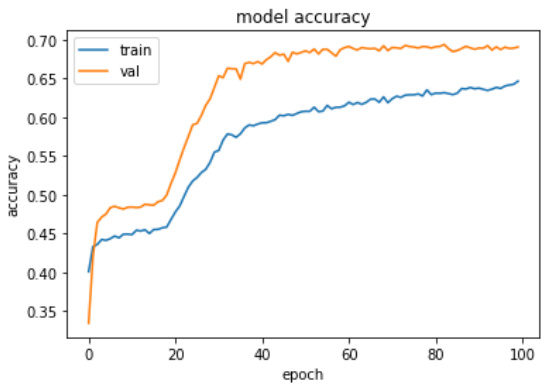
- Adam optimizer com parâmetros standard revelou-se melhor do que todos os SGD que experimentei. Optimizer SGD experimentado tinha cerca de 0.001 learning rate, 0.9 momentum, às vezes com decay outra vezes sem decay (se usasse decay ajustava o learning rate inicial para 0.1/0.01 dependendo do quão grande fosse o decay).

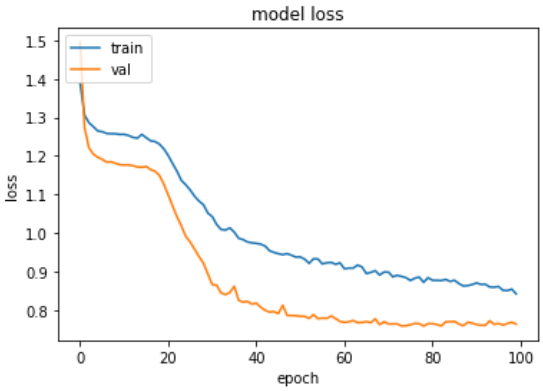
O modelo usado foi:



Comecei com uma layer convolucional, seguida de batchnormalization, seguido de SpatialDropout e depois maxpooling. Repeti estas 4 layers por 4x. De seguida apliquei as camadas recorrentes (LSTM), utilizei duas, ambas com 128 unidades e ambas com recurrent_dropout = 0.2. Por fim foram usadas as mesmas últimas layers que na convolucional, seguindo o mesmo raciocínio, com contas adaptadas ao modelo.

<u>Os resultados obtidos foram os seguintes:</u>



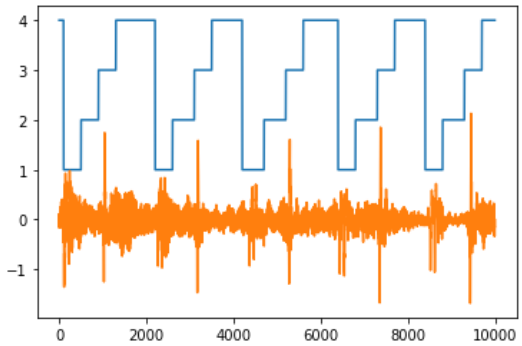


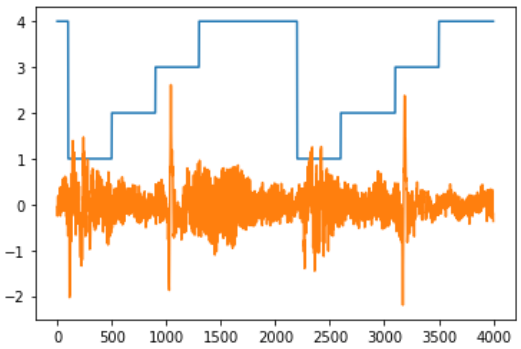

```
val accuracy:
0.6938991546630859
accuracy:
0.6465761065483093
val loss:
0.7581422924995422
train loss:
0.8416377902030945
```

Com valores de treino:

```
286/286 [==============================] - 48s 166ms/step - loss: 0.8540 - sparse_categorical_accuracy: 0.6405
Loss: 0.8539711833000183
Accuracy: 0.6404773592948914
```

Previsões:





4) Transformer

Para a rede recorrente foi usado:

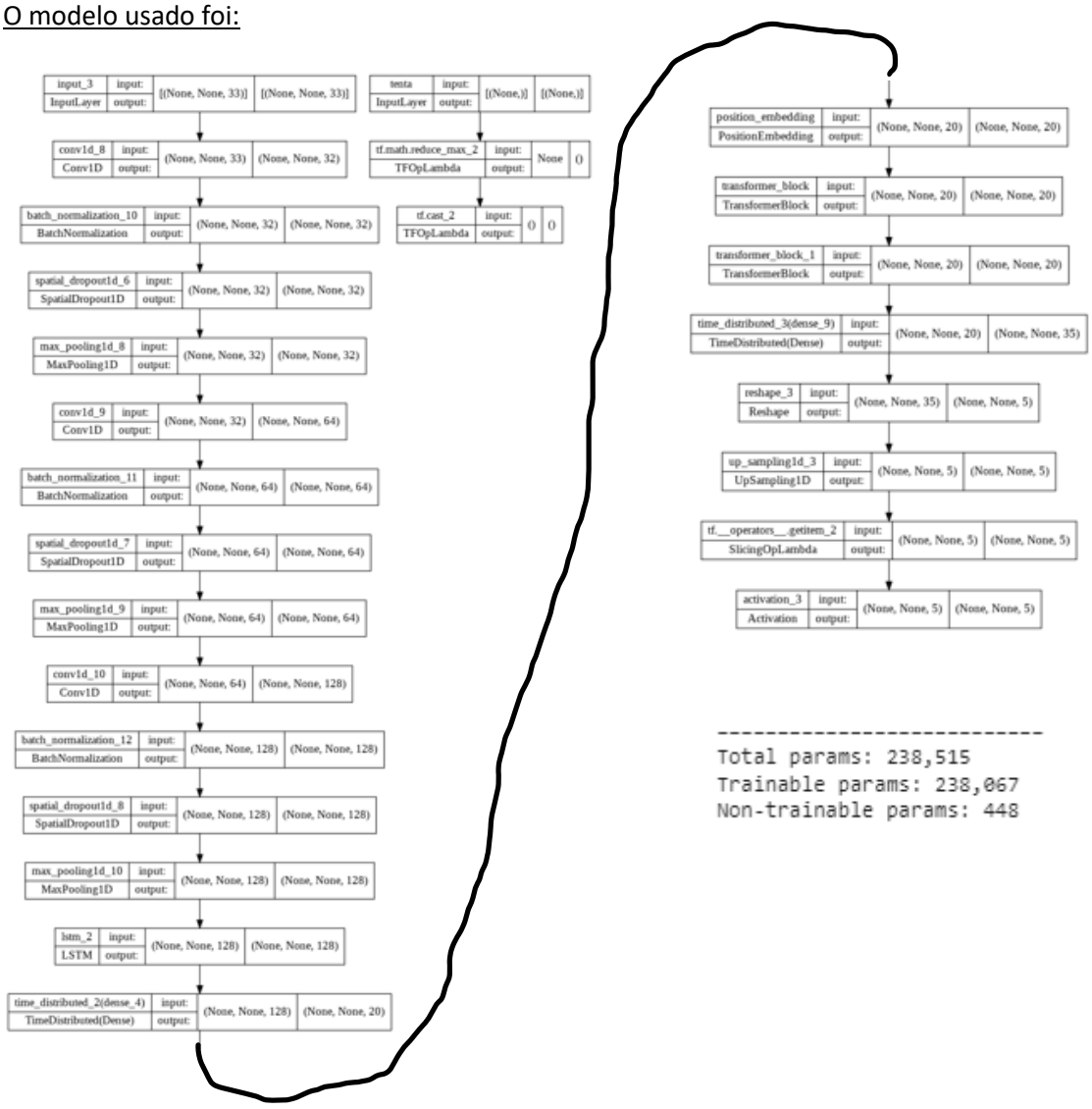
- Todos os ficheiros para treino, validação e teste, com tamanho variável;

- Short-time Fourier Transform com frame_length 64 e frame_step 32, torna o treino significativamente mais lento mas teoricamente melhora generalização melhorando os resultados obtidos;

- um batch size de 50, com 50 passos por época (50 x 50=2500) para garantir também uma maior generalização

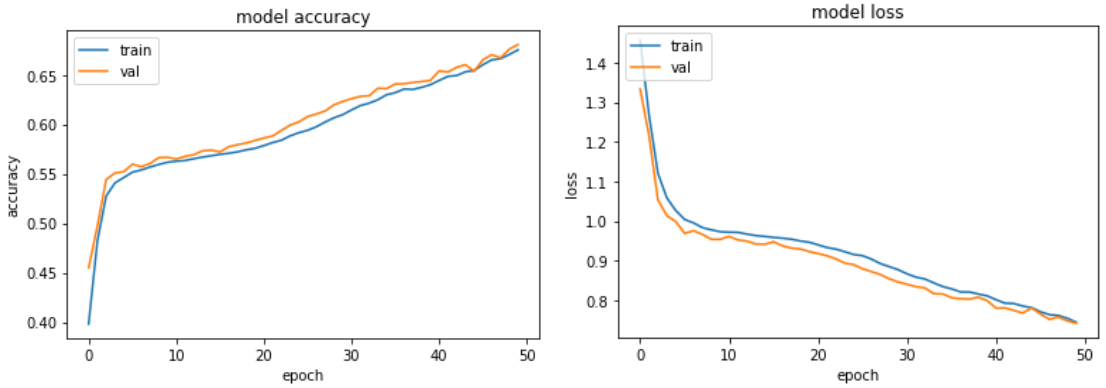
- Adam optimizer com parâmetros standard revelou-se melhor do que todos os SGD que experimentei. Optimizer SGD experimentado tinha cerca de 0.001 learning rate, 0.9 momentum, às vezes com decay outra vezes sem decay (se usasse decay ajustava o learning rate inicial para 0.1/0.01 dependendo do quão grande fosse o decay).

O modelo usado foi:



Foi usado o mesmo modelo da rede recorrente (mas com as 4 layers repetidas apenas 3 vezes) com as layers adicionais típicas de transformer. Ou seja, depois da camada LSTM foi usada uma layer timedistributed com dense dentro com o numero de camadas referente à embedded dimension. Seguindo de PositionEmbedding + dois blocos Transformer com embedded dimension, heads=6 e feedfoward dimension = 16. Por ultimo foram usadas as mesmas camadas finais tal como na convolucional e na recorrente

<u>Os resultados obtidos foram os seguintes:</u>



Não consegui correr da época 0 até 150 no colab apesar das várias tentativas acabava sempre por desconectar, optei por usar o model.fit noutra linha e corria 50 epocas de cada vez. O resultado referente às 150 epocas foram os seguintes:

```
val accuracy:
0.8372830748558044
accuracy:
0.8306281566619873
val loss:
0.41371309757232666
train loss:
0.4056330621242523
```

Com valores de treino:

```
286/286 [==============================] - 13s 43ms/step - loss: 0.9451 - sparse_categorical_accuracy: 0.7312
Loss: 0.9450841546058655
Accuracy: 0.7311709523200989
```

Previsões: