Gloria Miori

DEEP LEARNING ASSIGNMENT

Models:

NB: for all models I used:

- *'adam'* as optimizer:

```
model.compile(optimizer='adam',
      loss='sparse_categorical_crossentropy',
      metrics=['accuracy'])
```

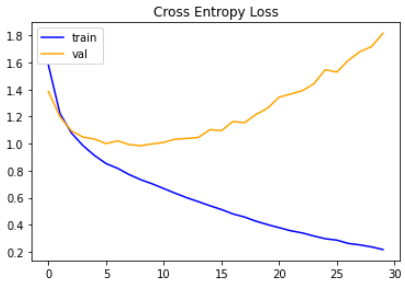
- batch_size = 128 (if not differently specified)
- epochs = 30 (except when EarlyStopping was used, of course)

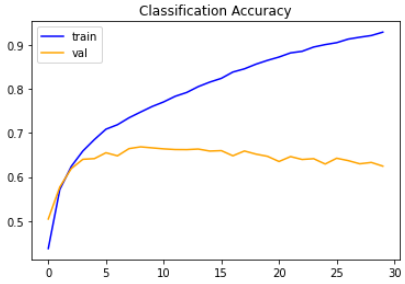
1) Starting with a basic model, very shallow: 2 convolutions and a MaxPooling. A flatten layer, followed directly by the 10 nodes output layer with activation function softmax:

```
[2]
    model = ks.Sequential()

    model.add(ks.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)))
    model.add(ks.layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.MaxPooling2D((2, 2)))

    model.add(ks.layers.Flatten()) # when exiting the Convolution, you have to do a Flatten, because FullDense layers expect arrays, not matrices
    model.add(ks.layers.Dense(10, activation='softmax'))
```

```
model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 10)                81930

=================================================================
Total params: 92,074
Trainable params: 92,074
Non-trainable params: 0
_________________________________________________________________
```

This model overfit very quickly the train data but does not work well on validation and test set. Entropy loss function increase quickly for validation set and accuracy does not improve:






```
_, acc = model.evaluate(x_test, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
```

```
> 62.070
```

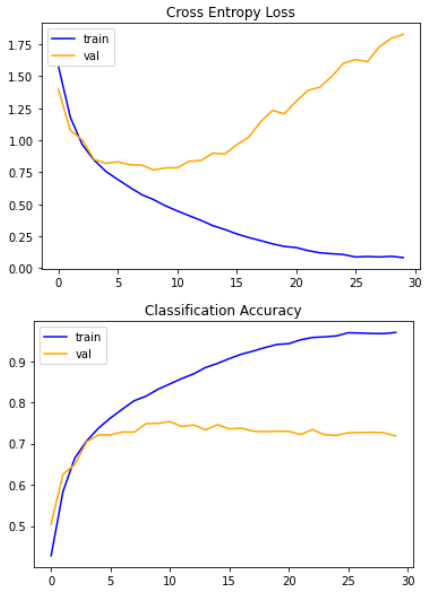
2) Adding 2 more convolution layers with an increasing number of nodes to the model:

```
[2]
    model = ks.Sequential()

    model.add(ks.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)))
    model.add(ks.layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.MaxPooling2D((2, 2)))

    model.add(ks.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.MaxPooling2D((2, 2)))

    model.add(ks.layers.Flatten()) # when exiting the Convolution, you have to do a Flatten, because FullDense layers expect arrays, not matrices
    model.add(ks.layers.Dense(10, activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 10)                40970

=================================================================
Total params: 106,538
Trainable params: 106,538
Non-trainable params: 0
_________________________________________________________________
```

This network starts working better than the previous one on the validation and test dataset but still overfit the training dataset and the validation and test accuracy is still low:




```
[14] _, acc = model.evaluate(x_test, y_test, verbose=0)
     print('> %.3f' % (acc * 100.0))
```

```
> 71.510
```

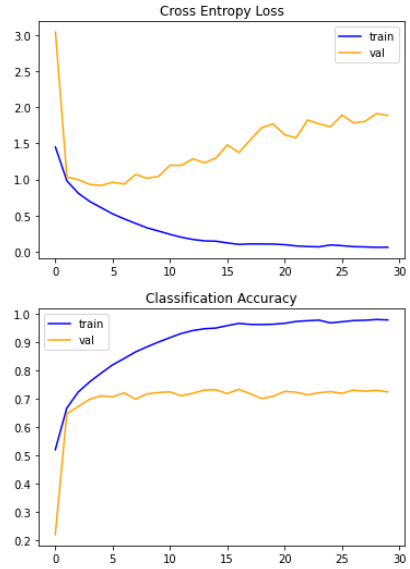
3) Adding Batch Normalization to prevent/reduce overfitting:

```
[3]
    model = ks.Sequential()

    model.add(ks.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.MaxPooling2D((2, 2)))

    model.add(ks.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.MaxPooling2D((2, 2)))

    model.add(ks.layers.Flatten()) # when exiting the Convolution, you have to do a Flatten, because FullDense layers expect arrays, not matrices
    model.add(ks.layers.Dense(10, activation='softmax'))
```

```
[4] model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 batch_normalization_1 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 batch_normalization_2 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 batch_normalization_3 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 10)                40970

=================================================================
Total params: 107,306
Trainable params: 106,922
Non-trainable params: 384
_________________________________________________________________
```

Not really improving the results for the validation and test dataset, and it is still overfitting:



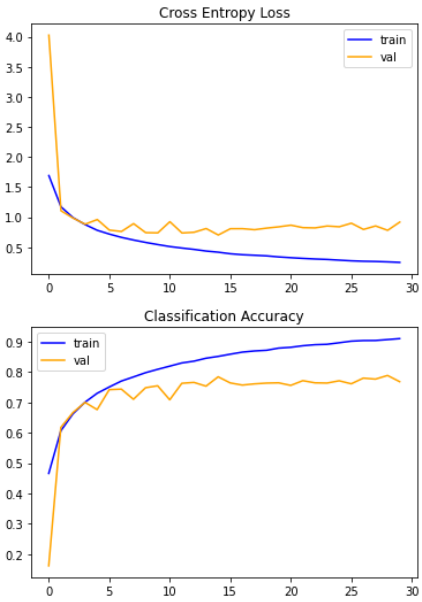

```
[15] _, acc = model.evaluate(x_test, y_test, verbose=0)
     print('> %.3f' % (acc * 100.0))

     > 71.150
```

4) Add a Dropout of 20% of neurons to reduce overfitting:

```
[2]
    model = ks.Sequential()

    model.add(ks.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.MaxPooling2D((2, 2)))
    model.add(ks.layers.Dropout(0.2))

    model.add(ks.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.MaxPooling2D((2, 2)))
    model.add(ks.layers.Dropout(0.2))

    model.add(ks.layers.Flatten()) # when exiting the Convolution, you have to do a Flatten, because FullDense layers expect arrays, not matrices
    model.add(ks.layers.Dense(10, activation='softmax'))
```

```
[3] model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 batch_normalization_1 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 dropout (Dropout)           (None, 16, 16, 32)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 batch_normalization_2 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 batch_normalization_3 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 8, 8, 64)          0

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 10)                40970

=================================================================
Total params: 107,306
Trainable params: 106,922
Non-trainable params: 384
_________________________________________________________________
```

It is getting better, but still overfitting and low accuracy for validation/tests. I would need a deeper network.




```
[14] _, acc = model.evaluate(x_test, y_test, verbose=0)
     print('> %.3f' % (acc * 100.0))

     > 75.750
```

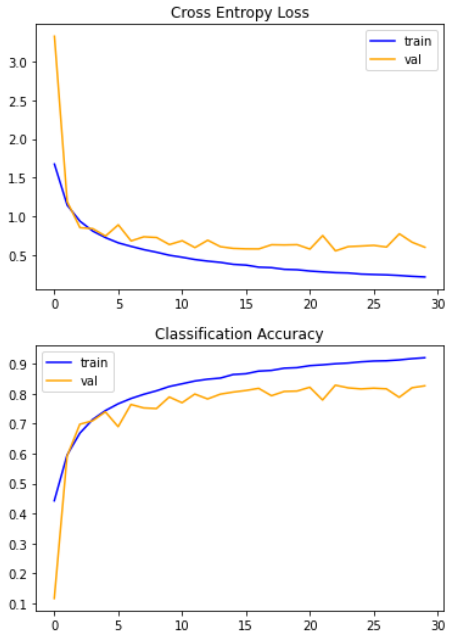
5) Adding 2 additional convolution layers with the same number of neurons. Keeping batch normalization and Dropout at 20%:

```
[2]
    model = ks.Sequential()

    model.add(ks.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.MaxPooling2D((2, 2)))
    model.add(ks.layers.Dropout(0.2))

    model.add(ks.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.MaxPooling2D((2, 2)))
    model.add(ks.layers.Dropout(0.2))

    model.add(ks.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.MaxPooling2D((2, 2)))
    model.add(ks.layers.Dropout(0.2))

    model.add(ks.layers.Flatten()) # when exiting the Convolution, you have to do a Flatten, because FullDense layers expect arrays, not matrices
    model.add(ks.layers.Dense(10, activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896

batch_normalization (BatchN  (None, 32, 32, 32)        128
ormalization)

conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248

batch_normalization_1 (Batc  (None, 32, 32, 32)        128
hNormalization)

max_pooling2d (MaxPooling2D  (None, 16, 16, 32)        0
)

dropout (Dropout)            (None, 16, 16, 32)        0

conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496

batch_normalization_2 (Batc  (None, 16, 16, 64)        256
hNormalization)

conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928

batch_normalization_3 (Batc  (None, 16, 16, 64)        256
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)          0
2D)

dropout_1 (Dropout)          (None, 8, 8, 64)          0

conv2d_4 (Conv2D)            (None, 8, 8, 64)          36928

batch_normalization_4 (Batc  (None, 8, 8, 64)          256
hNormalization)

conv2d_5 (Conv2D)            (None, 8, 8, 64)          36928

batch_normalization_5 (Batc  (None, 8, 8, 64)          256
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 4, 4, 64)          0
2D)

dropout_2 (Dropout)          (None, 4, 4, 64)          0

flatten (Flatten)            (None, 1024)              0

dense (Dense)                (None, 10)                10250

=================================================================
Total params: 150,954
Trainable params: 150,314
Non-trainable params: 640
_________________________________________________________________
```

It is still overfitting, but the validation/test accuracy is getting better:




```
[14] _, acc = model.evaluate(x_test, y_test, verbose=0)
     print('> %.3f' % (acc * 100.0))
```

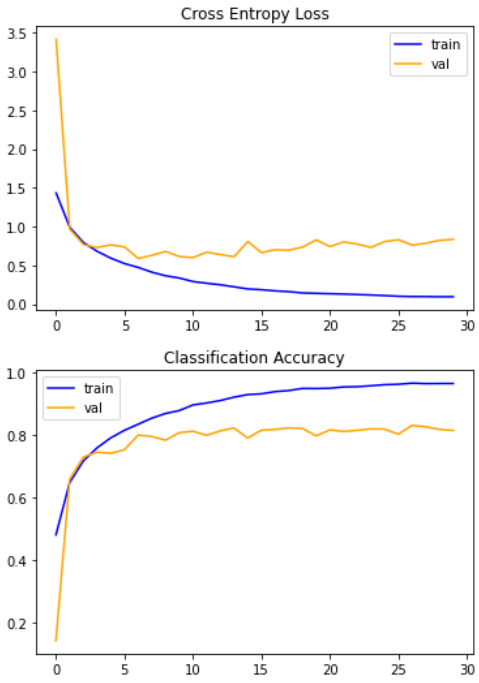
```
> 81.110
```

6) I will increase the number of neurons for the last 2 convolutions and adding dense layers (with decreasing neurons) before arriving at the final dense output layer of 10 nodes:

```
[2]
model = ks.Sequential()

model.add(ks.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Flatten()) # when exiting the Convolution, you have to do a Flatten, because FullDense layers expect arrays, not matrices
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dense(64, activation='relu'))
model.add(ks.layers.Dense(10, activation='softmax'))
```

```
model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 batch_normalization_1 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 dropout (Dropout)           (None, 16, 16, 32)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 batch_normalization_2 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 batch_normalization_3 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 8, 8, 64)          0

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 batch_normalization_4 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 batch_normalization_5 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 dropout_2 (Dropout)         (None, 4, 4, 128)         0

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 128)               262272

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 559,978
Trainable params: 559,082
Non-trainable params: 896
_________________________________________________________________
```

It does not improve the test/validation accuracy:




```
[14] _, acc = model.evaluate(x_test, y_test, verbose=0)
     print('> %.3f' % (acc * 100.0))

     > 80.720
```

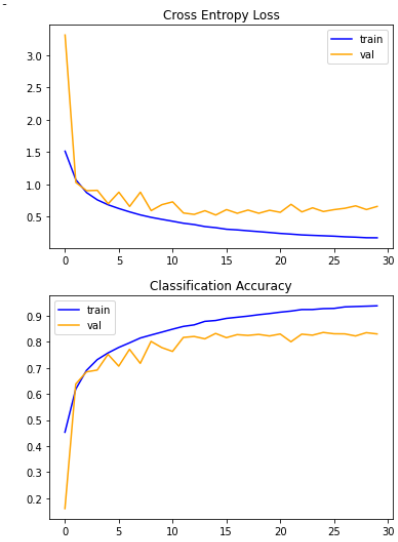
7) Let´s try than increasing the percentage of dropout as we increase the number of neurons (final model architecture):
   Six convolution layers neural network, all with padding and activation function relu. I kept the padding since the edge of the picture can be important for many of them. After each convolution I do a batch normalization to reduce overfitting. After 2 convolutions I use MaxPooling. I use an increase Dropout to reduce overfitting (0.2,0.3,0.4) since the number of neurons is as well increasing. All of this followed by a flatten layer and then different dense layers reducing the number of nodes to arrive at the final 10 nodes output dense layer (10 classes) with Softmax activation:

```
[2]
    model = ks.Sequential()

    model.add(ks.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.MaxPooling2D((2, 2)))
    model.add(ks.layers.Dropout(0.2))

    model.add(ks.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.MaxPooling2D((2, 2)))
    model.add(ks.layers.Dropout(0.3))

    model.add(ks.layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(ks.layers.BatchNormalization())
    model.add(ks.layers.MaxPooling2D((2, 2)))
    model.add(ks.layers.Dropout(0.4))

    model.add(ks.layers.Flatten()) # when exiting the Convolution, you have to do a Flatten, because FullDense layers expect arrays, not matrices
    model.add(ks.layers.Dense(128, activation='relu'))
    model.add(ks.layers.Dense(64, activation='relu'))
    model.add(ks.layers.Dense(10, activation='softmax'))
```

```
model.summary()
```

```
conv2d (Conv2D)              (None, 32, 32, 32)        896

batch_normalization (BatchN  (None, 32, 32, 32)        128
ormalization)

conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248

batch_normalization_1 (Batc  (None, 32, 32, 32)        128
hNormalization)

max_pooling2d (MaxPooling2D  (None, 16, 16, 32)        0
)

dropout (Dropout)            (None, 16, 16, 32)        0

conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496

batch_normalization_2 (Batc  (None, 16, 16, 64)        256
hNormalization)

conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928

batch_normalization_3 (Batc  (None, 16, 16, 64)        256
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)          0
2D)

dropout_1 (Dropout)          (None, 8, 8, 64)          0

conv2d_4 (Conv2D)            (None, 8, 8, 128)         73856

batch_normalization_4 (Batc  (None, 8, 8, 128)         512
hNormalization)

conv2d_5 (Conv2D)            (None, 8, 8, 128)         147584

batch_normalization_5 (Batc  (None, 8, 8, 128)         512
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)         0
2D)

dropout_2 (Dropout)          (None, 4, 4, 128)         0

flatten (Flatten)            (None, 2048)              0

dense (Dense)                (None, 128)               262272

dense_1 (Dense)              (None, 64)                8256

dense_2 (Dense)              (None, 10)                650

=================================================================
Total params: 559,978
Trainable params: 559,082
Non-trainable params: 896
_________________________________________________________________
```

It overfits later and improve the test/validation results:



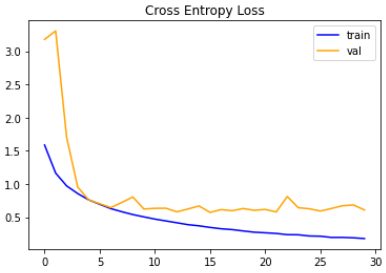

```
[14] _, acc = model.evaluate(x_test, y_test, verbose=0)
     print('> %.3f' % (acc * 100.0))
```

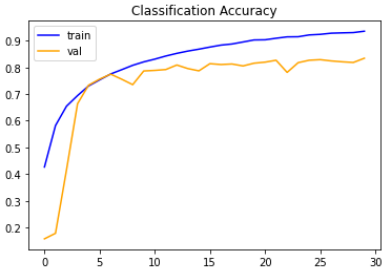
```
> 81.910
```

8) The same model as above in 7) but I have increased the batch size now:

```
history = model.fit(x_train, y_train, epochs=30,
          use_multiprocessing=False, batch_size= 256,
          validation_data=(x_val, y_val)
```

Increasing the validation/test results and it overfits later. It could be as well that the improvements are just related to random number differences.



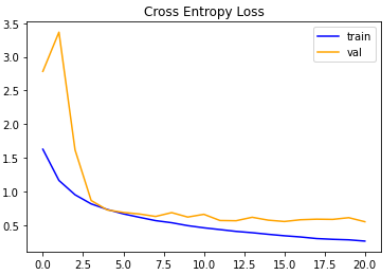


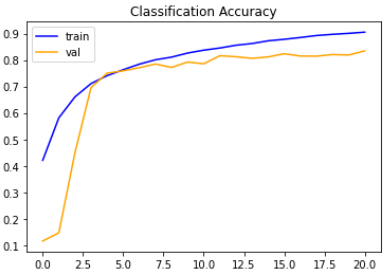
,

```
[15] _, acc = model.evaluate(x_test, y_test, verbose=0)
     print('> %.3f' % (acc * 100.0))

> 82.930
```

9) Maybe now we want to stop before overfitting: exactly same model as before (still batch_size = 256) but with EarlyStopping with patience = 5 for the cross entropy loss:

```
callback_val_loss = EarlyStopping(monitor="val_loss", min_delta=0.
01, patience=5, mode='auto')
```






```
[15] _, acc = model.evaluate(x_test, y_test, verbose=0)
     print('> %.3f' % (acc * 100.0))

> 82.490
```

The model seems to be already overfitting, we should have probably stop before (the train and validation dataset are already diverging in entropy and accuracy).

10) Same model architecture as in 7, 8 and 9, but using data augmentation and Early Stopping. I relaxed the EarlyStopping parameter patience since I have more data and the model is learning at a much “slower” pace. As well increasing the batch_size since I have more data now and the batch size influence how “noisy” the model results are.

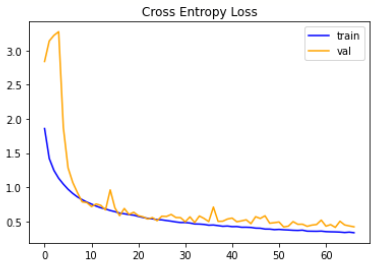
`loss='sparse_categorical_crossentropy' needed to be changed to 'categorical_crossentropy' when doing data augmentation`

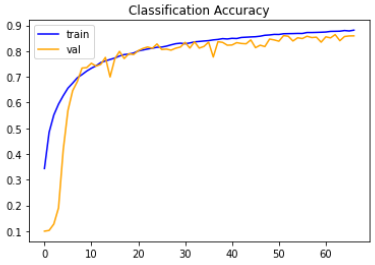
Other small changes were made in the code to be able to make data augmentation run. Anyway, the notebook with the final model and data augmentation is the one in the attached Jupiter notebook.

After different model run with epoch = 150 and evaluating when the model started overfitting, I found the “sweet spot” with the following EarlyStopping parameters (that I keep in the final notebook):

```
#training

callback_val_loss = EarlyStopping(monitor="val_loss", mode = 'min', patience=15, min_delta = 0.01)

history = model.fit(datagen.flow(x_train, y_train, batch_size=516),
                    epochs=150,validation_data=(x_val,y_val),
                    callbacks =callback_val_loss)
```

The model with data augmentation has better results on the test data, without overfitting. Now, it looks like the model with data augmentation stops when the train and validation loss and accuracy start to diverge:






```
_, acc = model.evaluate(x_test, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
```

```
> 85.420
```