Conv2D Parameters and Default Values

Wednesday, October 10, 2018 10:54 AM

Conv2D layer API Default Parameter Values

keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, dilation_rate=(1, 1), activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None)

Explaination from Official Documentation

- **filters**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size**: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides**: An integer or tuple/list of 2 integers, **specifying the strides of the convolution along the height and width**. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.
- **padding**: one of "valid" or "same" (**case-insensitive**). Note that "same" is slightly inconsistent across backends with strides != 1, as described here
- **data_format**: A string, one of "channels_last" or "channels_first". The ordering of the dimensions in the inputs. "channels_last" corresponds to inputs with shape (batch, height, width, channels) while "channels_first" corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".
- **dilation_rate**: an integer or tuple/list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.
- **activation**: Activation function to use (see activations). If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).
- **use_bias**: Boolean, whether the layer uses a bias vector.
- **kernel_initializer**: Initializer for the kernel weights matrix (see initializers).
- **bias_initializer**: Initializer for the bias vector (see initializers).
- **kernel_regularizer**: Regularizer function applied to the kernel weights matrix (see regularizer).
- **bias_regularizer**: Regularizer function applied to the bias vector (see regularizer).
- **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation"). (see regularizer).
- **kernel_constraint**: Constraint function applied to the kernel matrix (see constraints).
- **bias_constraint**: Constraint function applied to the bias vector (see constraints).

From <https://keras.io/layers/convolutional/>

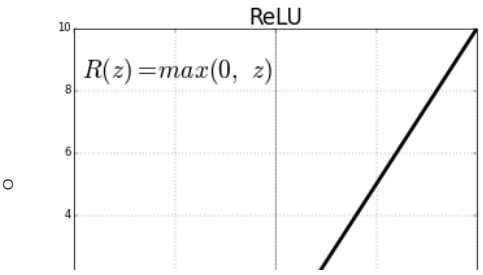
Filters

- "+": more weights, higher possible overfitting
- "-": less weights, underfitting, slowly converge

Kernel_size

Normally use 3, 5, 7
It's better to use small kernel size with more layers rather than large kernel size with less layers. The reason is related to **receptive field.**

Activation functions

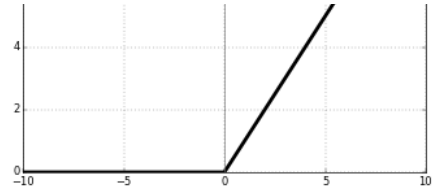
- Inside and outside the conv layers
- Choises:
  - Relu
  - 

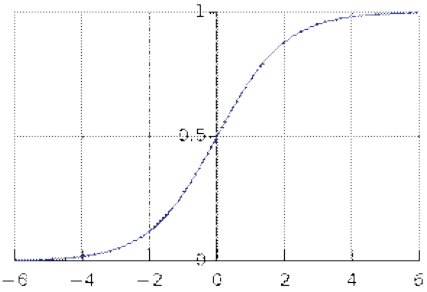
Kernel_initializer: glorot_normal
Bias_initializer: zeros

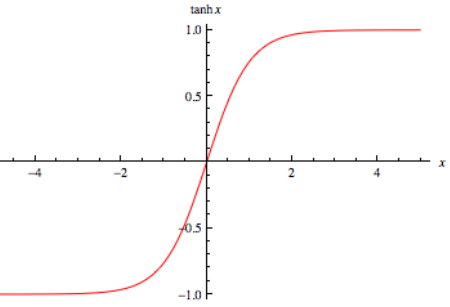
Choices:
zeros, ones, constant, RandomNormal, RandomUniform, TruncatedNormal, VarianceScaling, Orthogonal, Identity, lecun_uniform, glorot_normal, glorot_uniform, he_nromal, lecun_normal, he_uniform

kernel_regularizer, bias_regularizer, activity_regularizer

Regularizers allow to apply penalties on layer parameters or layer activity during

- 

- Sigmoid
- 

- Tanh
- 

- Relu is a better choice than sigmoid or tanh
  This is because relu has two benefits:
  sparsity and lateral inhibition which can reduce the likelihood of vanishing gradient.

kernel_regularizer, bias_regularizer, activity_regularizer

Regularizers allow to apply penalties on layer parameters or layer activity during optimization.

When model get overfitted, use regularizer to improve the ability of generlization of your model.

Example:

```
from keras import regularizers
model.add(Dense(64, input_dim=64,
                kernel_regularizer=regularizers.l2(0.01),
                activity_regularizer=regularizers.l1(0.01)))
```

Available penalties:

keras.regularizers.l1(0.)
keras.regularizers.l2(0.)
keras.regularizers.l1_l2(l1=0.01, l2=0.01)

From <https://keras.io/regularizers/>

Weights Initialiation

Wednesday, October 10, 2018 10:34 AM

The normal vs uniform init seem to be rather unclear in fact.
If we refer solely on the Glorot's and He's initializations papers, they both use a similar theoritical analysis: they find a good variance for the distribution from which the initial parameters are drawn. This variance is adapted to the activation function used and is derived without explicitly considering the type of the distribution. As such, their theorical conclusions hold for any type of distribution of the determined variance. In fact, in the Glorot paper, a uniform distribution is used whereas in the He paper it is a gaussian one that is chosen. The only "explaination" given for this choice in the He paper is:
Recent deep CNNs are mostly initialized by random weights drawn from Gaussian distributions
with a reference to AlexNet paper. It was indeed released a little later than Glorot's initialization but however there is no justificaion in it of the use of a normal distribution.
In fact, in a discussion on Keras issues tracker, they also seem to be a little confused and basically it could only be a matter of preference... (i.e. hypotetically Bengio would prefer uniform distribution whereas Hinton would prefer normal ones...) One the discussion, there is a small benchmark comparing Glorot initialization using a uniform and a gaussian distribution. In the end, it seems that the uniform wins but it is not really clear.
In the original ResNet paper, it only says they used a gaussian He init for all the layers, I was not able to find where it is written that they used a uniform He init for the first layer. (maybe you could share a reference to this?)
As for the use of gaussian init with Batch Normalization, well, with BN the optimization process is less sensitive to initialization thus it is just a convention I would say.

From <https://datascience.stackexchange.com/questions/13061/when-to-use-he-or-glorot-normal-initialization-over-uniform-init-and-what-are>

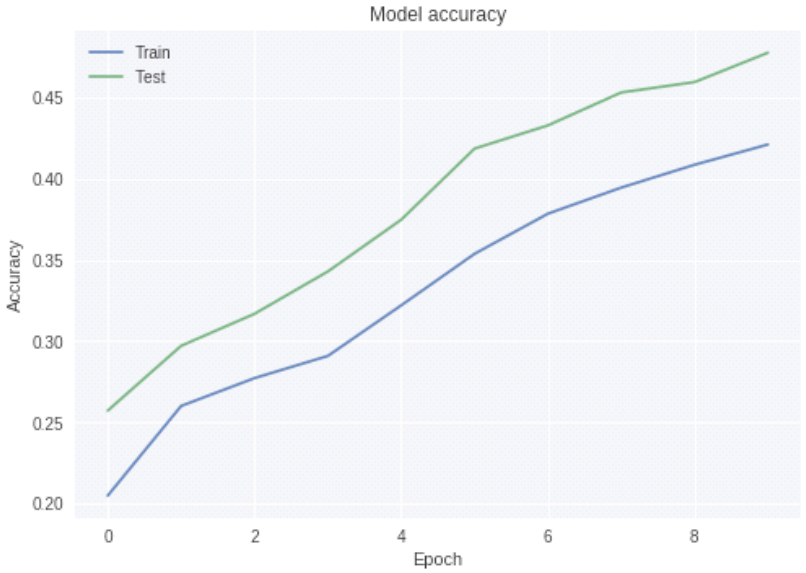
Ex1: Default model

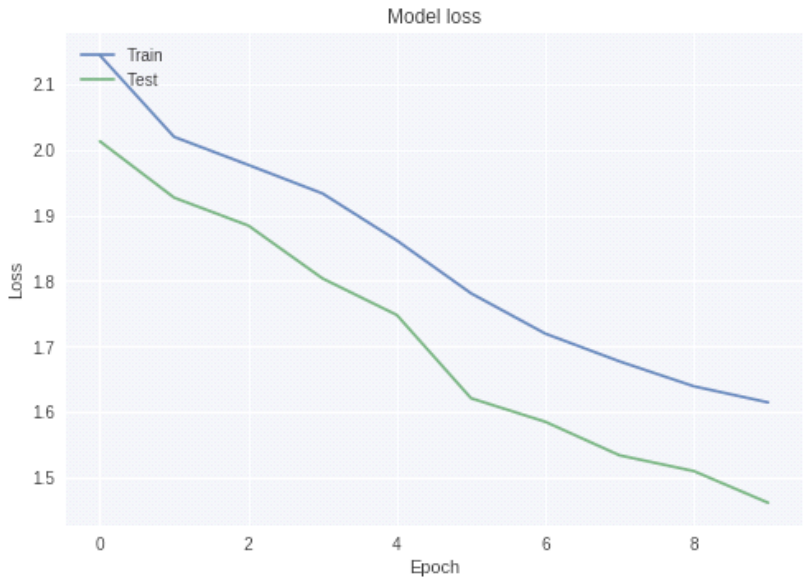
Tuesday, October 9, 2018 11:02 AM

Ex1:

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

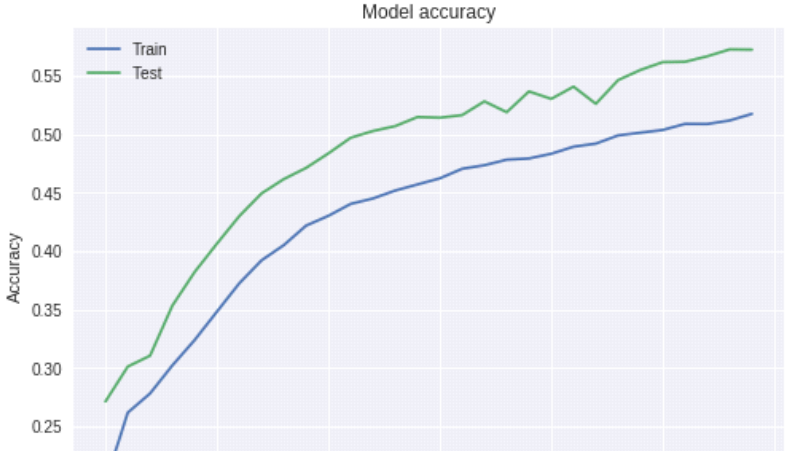
10 epochs



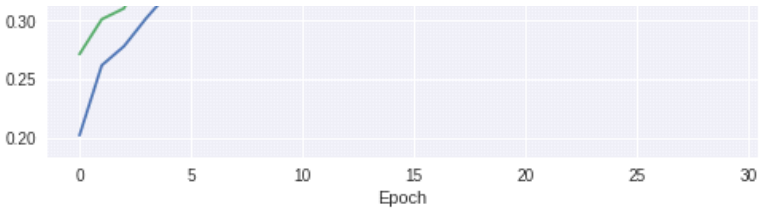


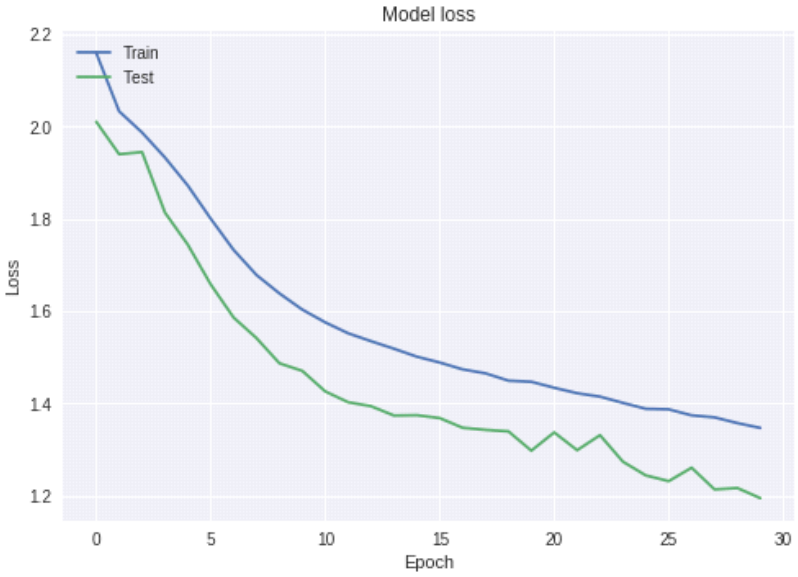

```
Epoch 1/100
1563/1563 [==============================] -
52s 34ms/step - loss: 1.9137 - acc: 0.2947 - val_loss:
1.5951 - val_acc: 0.4309
Epoch 2/100
1563/1563 [==============================] -
51s 33ms/step - loss: 1.6076 - acc: 0.4114 - val_loss:
1.5008 - val_acc: 0.4652
Epoch 3/100
1563/1563 [==============================] -
51s 32ms/step - loss: 1.4872 - acc: 0.4602 - val_loss:
1.4133 - val_acc: 0.5002
Epoch 4/100
1563/1563 [==============================] -
50s 32ms/step - loss: 1.4070 - acc: 0.4922 - val_loss:
1.2305 - val_acc: 0.5558
Epoch 5/100
1563/1563 [==============================] -
50s 32ms/step - loss: 1.3472 - acc: 0.5173 - val_loss:
1.2242 - val_acc: 0.5685
```

30 epochs



Epoch 5/100
1563/1563 [==============================] - 50s 32ms/step - loss: 1.3472 - acc: 0.5173 - val_loss: 1.2242 - val_acc: 0.5685
Epoch 6/100
1563/1563 [==============================] - 51s 32ms/step - loss: 1.2886 - acc: 0.5405 - val_loss: 1.1495 - val_acc: 0.5881
Epoch 7/100
1563/1563 [==============================] - 51s 32ms/step - loss: 1.2314 - acc: 0.5633 - val_loss: 1.0979 - val_acc: 0.6164
Epoch 8/100
1563/1563 [==============================] - 51s 33ms/step - loss: 1.1890 - acc: 0.5795 - val_loss: 1.0800 - val_acc: 0.6165
Epoch 9/100
1563/1563 [==============================] - 51s 33ms/step - loss: 1.1551 - acc: 0.5907 - val_loss: 1.0318 - val_acc: 0.6294
Epoch 10/100
1563/1563 [==============================] - 51s 32ms/step - loss: 1.1223 - acc: 0.6037 - val_loss: 0.9952 - val_acc: 0.6482
Epoch 11/100
1563/1563 [==============================] - 51s 32ms/step - loss: 1.0910 - acc: 0.6168 - val_loss: 0.9556 - val_acc: 0.6660
Epoch 12/100
1563/1563 [==============================] - 50s 32ms/step - loss: 1.0642 - acc: 0.6260 - val_loss: 0.9937 - val_acc: 0.6525
Epoch 13/100
1563/1563 [==============================] - 51s 32ms/step - loss: 1.0439 - acc: 0.6309 - val_loss: 0.9177 - val_acc: 0.6807
Epoch 14/100
1563/1563 [==============================] - 50s 32ms/step - loss: 1.0178 - acc: 0.6425 - val_loss: 0.9341 - val_acc: 0.6800
Epoch 15/100
1563/1563 [==============================] - 51s 33ms/step - loss: 1.0001 - acc: 0.6506 - val_loss: 0.9134 - val_acc: 0.6855
Epoch 16/100
1563/1563 [==============================] - 51s 33ms/step - loss: 0.9794 - acc: 0.6579 - val_loss: 0.8992 - val_acc: 0.6867
Epoch 17/100
1563/1563 [==============================] - 50s 32ms/step - loss: 0.9590 - acc: 0.6652 - val_loss: 0.8850 - val_acc: 0.6964
Epoch 18/100
1563/1563 [==============================] - 51s 32ms/step - loss: 0.9494 - acc: 0.6677 - val_loss: 0.8405 - val_acc: 0.7124
Epoch 19/100
1563/1563 [==============================] - 51s 33ms/step - loss: 0.9361 - acc: 0.6763 - val_loss: 0.8200 - val_acc: 0.7175
Epoch 20/100
1563/1563 [==============================] - 51s 33ms/step - loss: 0.9249 - acc: 0.6787 - val_loss:





0.8467 - val_acc: 0.7037
Epoch 21/100
1563/1563 [==============================] -
51s 33ms/step - loss: 0.9189 - acc: 0.6815 - val_loss:
0.7741 - val_acc: 0.7367

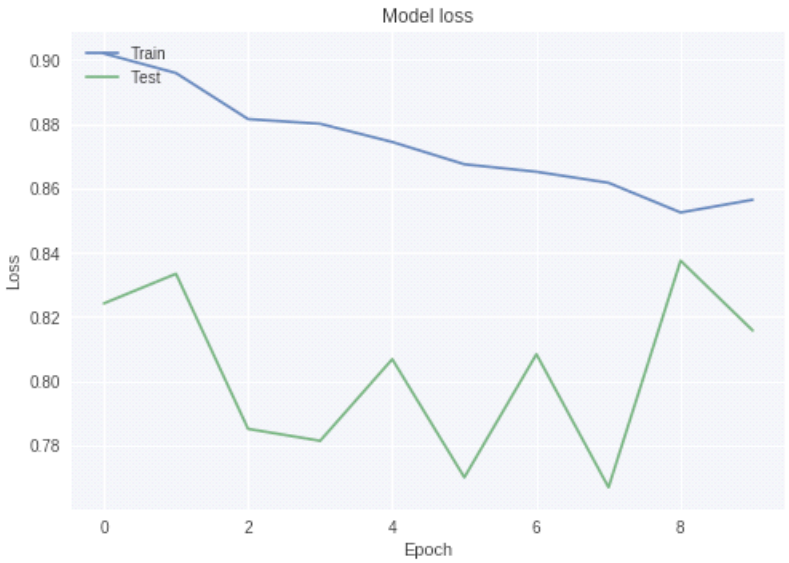
Ex2: Double the numbers of filters

Tuesday, October 9, 2018 11:27 AM

Double the numbers of filters

Ex2:

```
model = Sequential()
model.add(Conv2D(64, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```



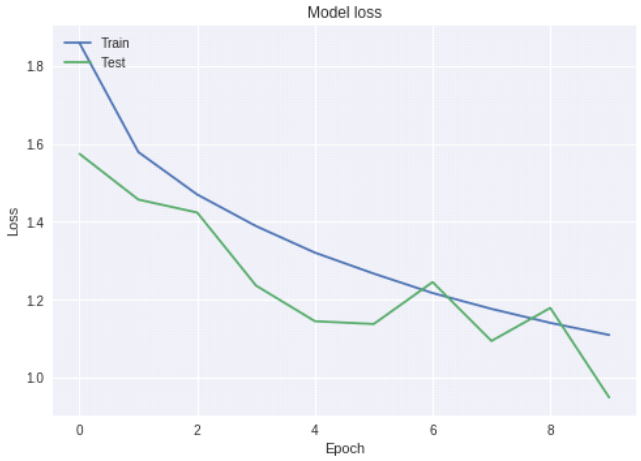
Ex3: Inside Relu

Tuesday, October 9, 2018 11:51 AM

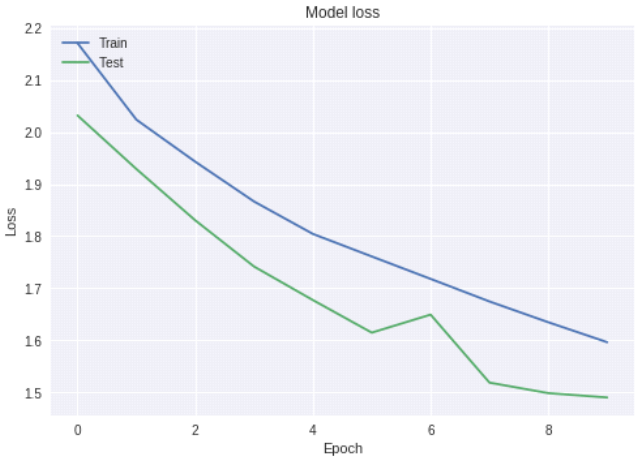
Use the activation functions inside conv2D layers

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:],activation='relu'))
#model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3),activation='relu'))
#model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same',activation='relu'))
#model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3),activation='relu'))
#model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.summary()
```

10 epochs



30 epochs


Model loss
Train
Test
Loss
Epoch
2.2
2.1
2.0
1.9
1.8
1.7
1.6
1.5
0
2
4
6
8

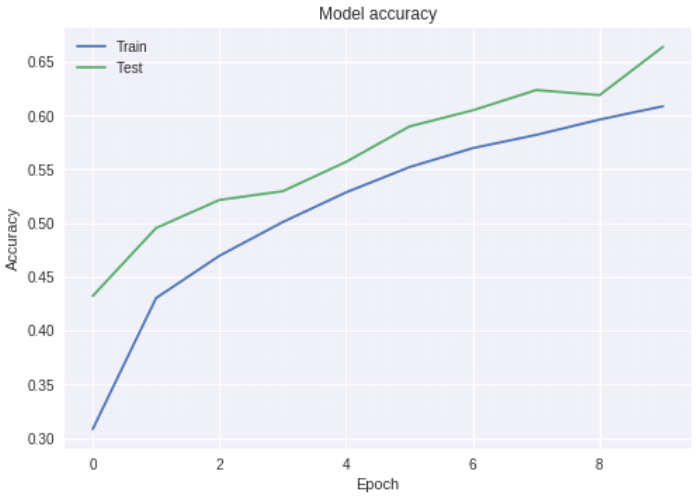
Ex4: Not using bias inside conv2D layers

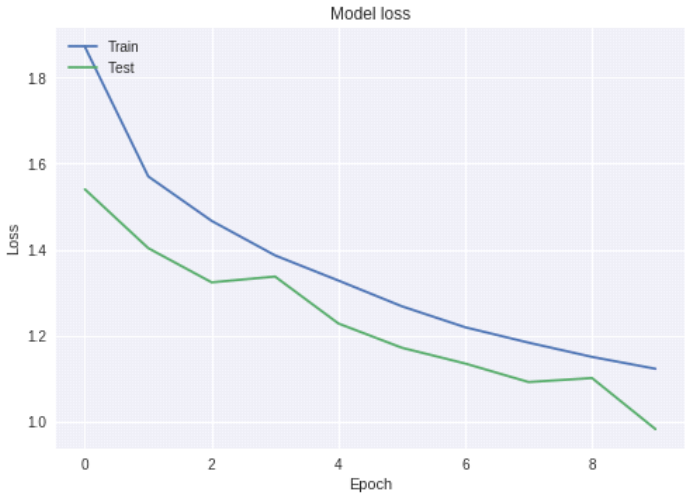
Tuesday, October 9, 2018 1:24 PM

Not using bias inside conv2D layers

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:], use_bias=False))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3), use_bias=False))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same', use_bias=False))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3), use_bias=False))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.summary()
```

10 epochs





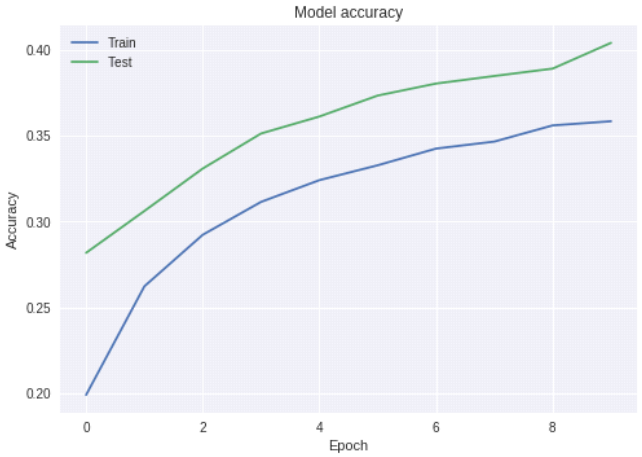
Ex5: Using half numbers of filters

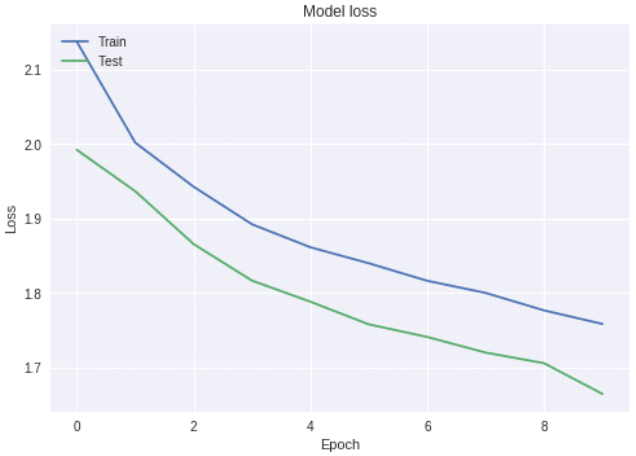
Tuesday, October 9, 2018 1:45 PM

With less filters

```
model = Sequential()
model.add(Conv2D(16, (3, 3), padding='same',
                 input_shape=x_train.shape[1:], use_bias=False))
model.add(Activation('relu'))
model.add(Conv2D(16, (3, 3), use_bias=False))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(32, (3, 3), padding='same', use_bias=False))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3), use_bias=False))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.summary()
```

10 epochs





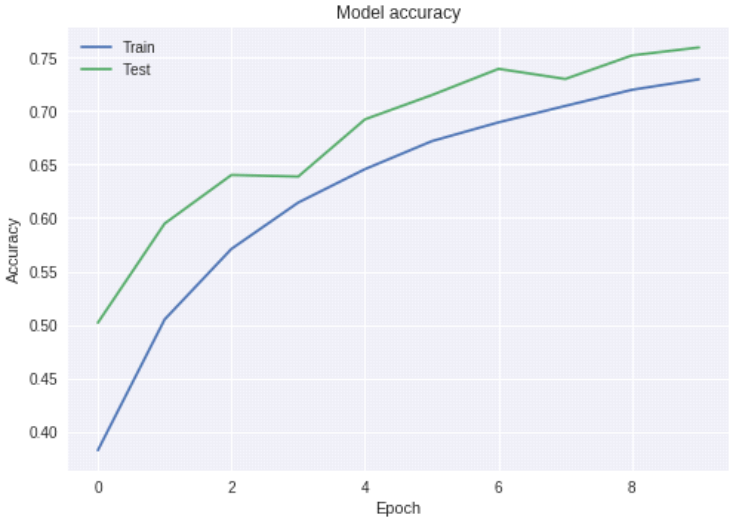
Ex6: Triple the numbers of filters

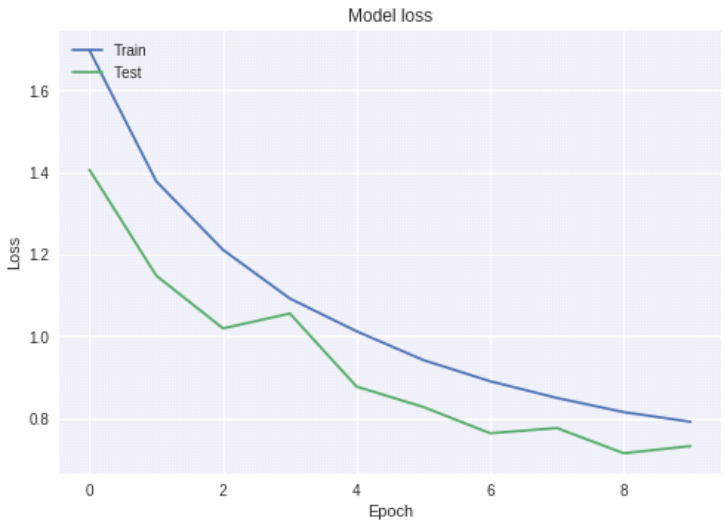
Wednesday, October 10, 2018 10:21 AM

Triple the numbers of filters

```
model = Sequential()
model.add(Conv2D(96, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(96, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(192, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(192, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.summary()
```

10 epochs





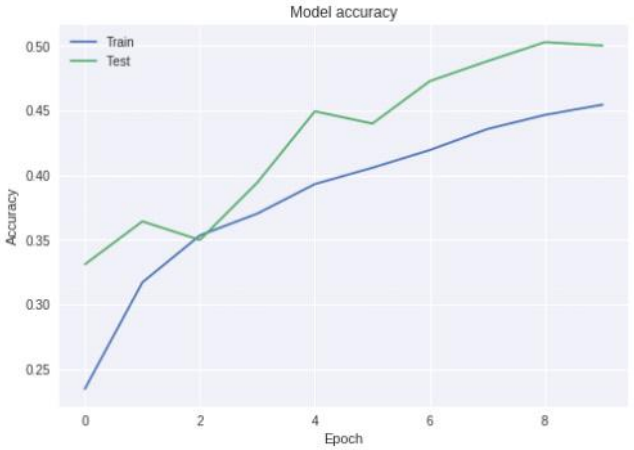
Ex7: Using He normal

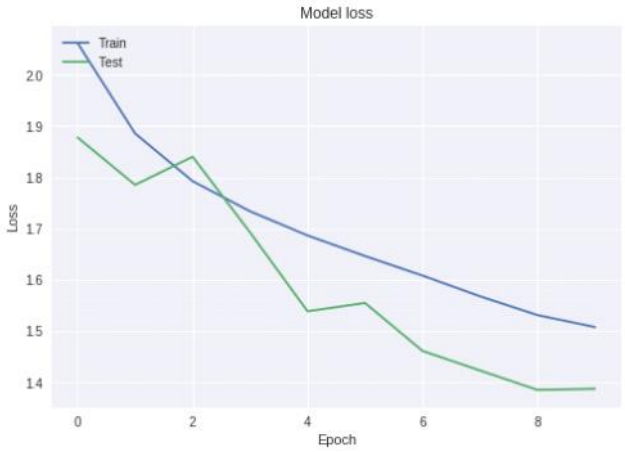
Wednesday, October 10, 2018 10:35 AM

Using He normal

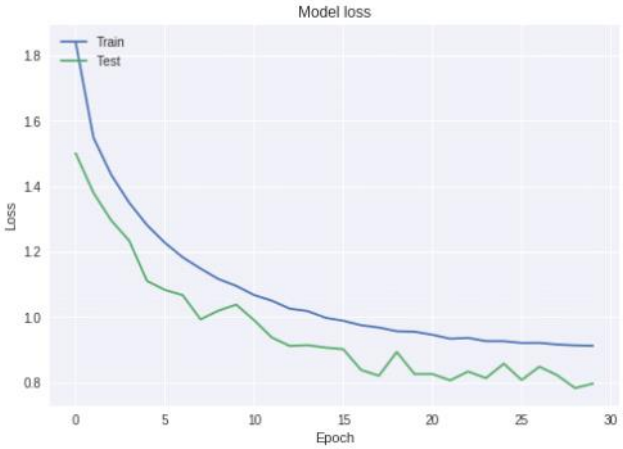
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',kernel_initializer='he_normal',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3),kernel_initializer='he_normal'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same',kernel_initializer='he_normal'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3),kernel_initializer='he_normal'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.summary()
```

10 epochs





30 epochs

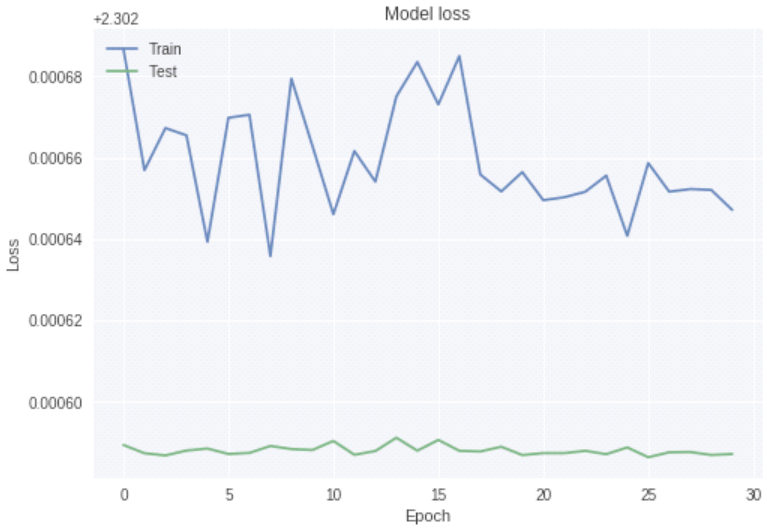

Model loss
Train
Test
Loss
Epoch
1.8
1.6
1.4
1.2
1.0
0.8
0
5
10
15
20
25
30

Ex8: Inside Sigmoid

Wednesday, October 10, 2018 10:39 AM

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:],activatic
model.add(Conv2D(32, (3, 3),activation='sigmoid'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same',activation='
model.add(Conv2D(64, (3, 3),activation='sigmoid'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.summary()
```

30 epochs

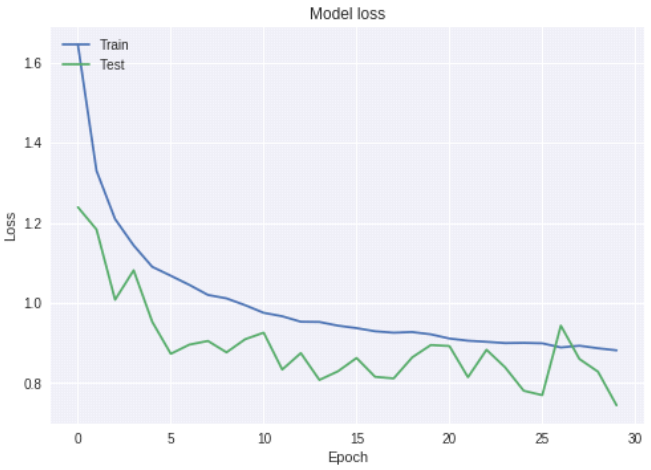


Ex9: Inside tanh

Wednesday, October 10, 2018 10:39 AM

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:],activation='
model.add(Conv2D(32, (3, 3),activation='tanh'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same',activation='tan
model.add(Conv2D(64, (3, 3),activation='tanh'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.summary()
```

30 epochs



Ex10: Using (5,5) kernel size

Wednesday, October 10, 2018 1:15 PM

```
model = Sequential()
model.add(Conv2D(32, (5, 5), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (5, 5)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (5, 5), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (5, 5)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.summary()
```

30 epochs