Experiments

Wednesday, October 10, 2018 2:50 PM

I have done many experiments to find the best model. Here I only put two of them because among all the experiments, most of them are not completely trained due to the concern of time and the early finding of something wrong.
The two experiments I put here are the almost-finshed models.
You can see the model in ex1 is underfitting, which means the model still lacks the ability of learning on training set, so I decided to reduce the dropout rates. Then there was ex2. The model in ex2 was still underfitting and after several rounds of experiments, the dropout rate was determined and I got my best model.

Ex1

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
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(BatchNormalization())
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
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(BatchNormalization())
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
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()
```

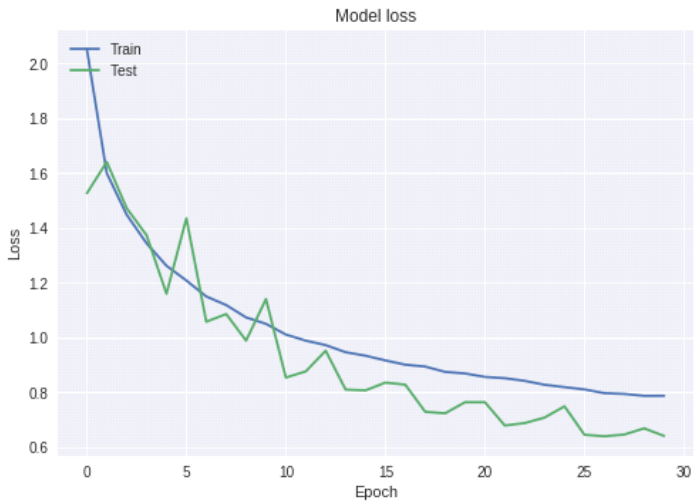
30 epochs

```
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)

model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

Ex2

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.20))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.20))

model.add(Conv2D(128, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.20))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.summary()
```

30 epochs




```
Using real-time data augmentation.
Epoch 1/30
1563/1563 [==============================] - 67s 43ms/step - loss: 1.8805 - acc: 0.3395 - val_loss: 1.4263 - val_acc: 0.4826
Epoch 2/30
1563/1563 [==============================] - 63s 40ms/step - loss: 1.4735 - acc: 0.4683 - val_loss: 1.3686 - val_acc: 0.5249
Epoch 3/30
1563/1563 [==============================] - 63s 40ms/step - loss: 1.3139 - acc: 0.5325 - val_loss: 1.3091 - val_acc: 0.5472
Epoch 4/30
1563/1563 [==============================] - 63s 40ms/step - loss: 1.2197 - acc: 0.5660 - val_loss: 1.2638 - val_acc: 0.5720
```

```
Epoch 5/30
1563/1563 [==============================] - 63s 41ms/step - loss: 1.1384 - acc: 0.5947 - val_loss: 1.0151 - val_acc: 0.6426
Epoch 6/30
1563/1563 [==============================] - 64s 41ms/step - loss: 1.0779 - acc: 0.6181 - val_loss: 1.0563 - val_acc: 0.6306
Epoch 7/30
1563/1563 [==============================] - 65s 42ms/step - loss: 1.0208 - acc: 0.6415 - val_loss: 0.9082 - val_acc: 0.6762
Epoch 8/30
1563/1563 [==============================] - 63s 40ms/step - loss: 0.9834 - acc: 0.6543 - val_loss: 0.8742 - val_acc: 0.6976
Epoch 9/30
1563/1563 [==============================] - 64s 41ms/step - loss: 0.9532 - acc: 0.6672 - val_loss: 0.8458 - val_acc: 0.7058
Epoch 10/30
1563/1563 [==============================] - 63s 41ms/step - loss: 0.9163 - acc: 0.6823 - val_loss: 0.8946 - val_acc: 0.6934
Epoch 11/30
1563/1563 [==============================] - 63s 40ms/step - loss: 0.8865 - acc: 0.6926 - val_loss: 0.9104 - val_acc: 0.6878
Epoch 12/30
1563/1563 [==============================] - 64s 41ms/step - loss: 0.8636 - acc: 0.7001 - val_loss: 0.8476 - val_acc: 0.7123
Epoch 13/30
1563/1563 [==============================] - 64s 41ms/step - loss: 0.8441 - acc: 0.7066 - val_loss: 0.7355 - val_acc: 0.7418
Epoch 14/30
1563/1563 [==============================] - 65s 42ms/step - loss: 0.8242 - acc: 0.7139 - val_loss: 0.7577 - val_acc: 0.7393
Epoch 15/30
1563/1563 [==============================] - 64s 41ms/step - loss: 0.8074 - acc: 0.7224 - val_loss: 0.7039 - val_acc: 0.7578
Epoch 16/30
1563/1563 [==============================] - 64s 41ms/step - loss: 0.7935 - acc: 0.7270 - val_loss: 0.7371 - val_acc: 0.7471
Epoch 17/30
1563/1563 [==============================] - 64s 41ms/step - loss: 0.7805 - acc: 0.7320 - val_loss: 0.7406 - val_acc: 0.7437
Epoch 18/30
1563/1563 [==============================] - 65s 41ms/step - loss: 0.7690 - acc: 0.7358 - val_loss: 0.6382 - val_acc: 0.7836
Epoch 19/30
1563/1563 [==============================] - 64s 41ms/step - loss: 0.7591 - acc: 0.7395 - val_loss: 0.6321 - val_acc: 0.7838
Epoch 20/30
1563/1563 [==============================] - 63s 40ms/step - loss: 0.7564 - acc: 0.7427 - val_loss: 0.6834 - val_acc: 0.7697
Epoch 21/30
1563/1563 [==============================] - 63s 40ms/step - loss: 0.7375 - acc: 0.7479 - val_loss: 0.6770 - val_acc: 0.7720
Epoch 22/30
1563/1563 [==============================] - 63s 40ms/step - loss: 0.7263 - acc: 0.7536 - val_loss: 0.6573 - val_acc: 0.7800
Epoch 23/30
1563/1563 [==============================] - 63s 41ms/step - loss: 0.7204 - acc: 0.7538 - val_loss: 0.6033 - val_acc: 0.7968
Epoch 24/30
1563/1563 [==============================] - 65s 41ms/step - loss: 0.7064 - acc: 0.7599 - val_loss: 0.6577 - val_acc: 0.7777
Epoch 25/30
1563/1563 [==============================] - 65s 41ms/step - loss: 0.7030 - acc: 0.7584 - val_loss: 0.6628 - val_acc: 0.7747
Epoch 26/30
1563/1563 [==============================] - 64s 41ms/step - loss: 0.6885 - acc: 0.7621 - val_loss: 0.6306 - val_acc: 0.7883
Epoch 27/30
1563/1563 [==============================] - 64s 41ms/step - loss: 0.6859 - acc: 0.7664 - val_loss: 0.6513 - val_acc: 0.7811
Epoch 28/30
1563/1563 [==============================] - 64s 41ms/step - loss: 0.6851 - acc: 0.7677 - val_loss: 0.5903 - val_acc: 0.8052
Epoch 29/30
1563/1563 [==============================] - 64s 41ms/step - loss: 0.6795 - acc: 0.7723 - val_loss: 0.6170 - val_acc: 0.7927
Epoch 30/30
1563/1563 [==============================] - 63s 40ms/step - loss: 0.6629 - acc: 0.7779 - val_loss: 0.7231 - val_acc: 0.7705
10000/10000 [==============================] - 3s 271us/step
Test loss: 0.723122650051117
Test accuracy: 0.7705
```

Best Model and Findings

Friday, October 12, 2018 3:35 PM

Findings

1. Training with CNN(and other deep learning models) can be very time-consuming on personal computers.
2. It is useful to add batch normalization to your CNN model. It can make the model more generalizable, preventing overfitting.
3. By controlling the dropout rate, we can control the learning ability of our model, which can be helpful when the neural network model is a little underfitting.
4. When turning the learning rate, we can multiply the learning rate by 3. for example, from 0.0001, then 0.0003, then 0.001, 0.003, 0.01, etc..
5. Reading related articles and papers is really important especially for beginners like us. Deep learning is quite based on experience, learn from others' experience can save you plenty of time when building and training your own model.
6. It is important to get familiar to those machine learning/deep learning packages.

Best Model

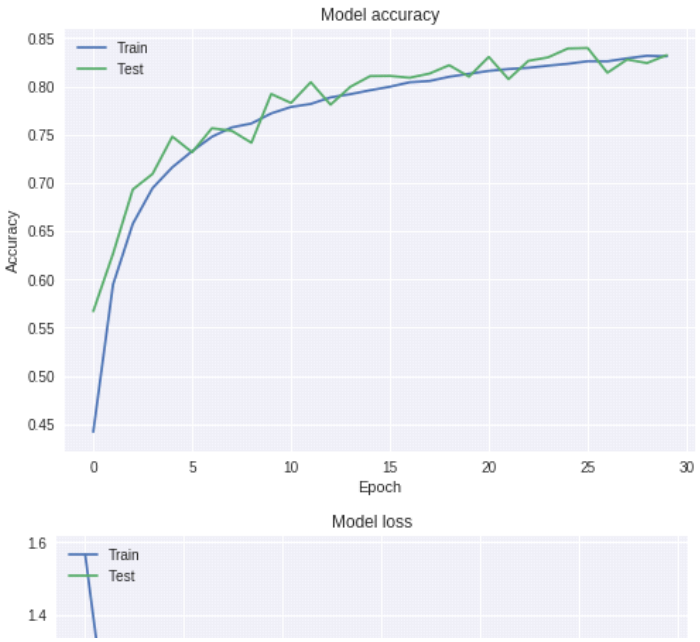
This is the best model I've got after many experiments. It is perfectly fit, without even a bit of underfitting nor overfitting and achieved 83% accuracy on testing set.

Measurement

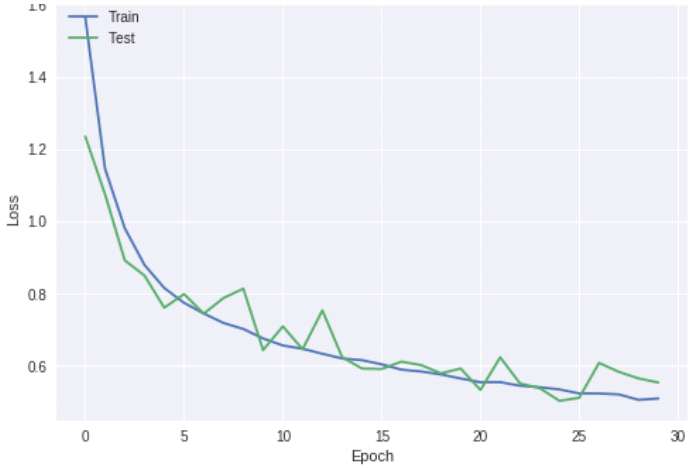
Loss: categorical_crossentropy
Metrics: accuracy

Test loss: 0.553475624370575 Test accuracy: 0.8323

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.1))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.1))

model.add(Conv2D(128, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3)))
```



```
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.1))


model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.summary()


# initiate RMSprop optimizer
opt = keras.optimizers.rmsprop(lr=0.0003, decay=1e-6)

# Let's train the model using RMSprop
model.compile(loss='categorical_crossentropy',
        optimizer=opt,
        metrics=['accuracy'])
```




```
Using real-time data augmentation.
Epoch 1/30 1563/1563 [==============================] - 62s 40ms/step - loss: 1.5649 - acc: 0.4421 - val_loss: 1.2340 - val_acc: 0.5671
Epoch 2/30 1563/1563 [==============================] - 58s 37ms/step - loss: 1.1453 - acc: 0.5946 - val_loss: 1.0748 - val_acc: 0.6266
Epoch 3/30 1563/1563 [==============================] - 58s 37ms/step - loss: 0.9806 - acc: 0.6577 - val_loss: 0.8915 - val_acc: 0.6932
Epoch 4/30 1563/1563 [==============================] - 58s 37ms/step - loss: 0.8785 - acc: 0.6946 - val_loss: 0.8492 - val_acc: 0.7093
Epoch 5/30 1563/1563 [==============================] - 58s 37ms/step - loss: 0.8150 - acc: 0.7162 - val_loss: 0.7605 - val_acc: 0.7480
Epoch 6/30 1563/1563 [==============================] - 58s 37ms/step - loss: 0.7740 - acc: 0.7329 - val_loss: 0.7981 - val_acc: 0.7317
Epoch 7/30 1563/1563 [==============================] - 58s 37ms/step - loss: 0.7444 - acc: 0.7478 - val_loss: 0.7439 - val_acc: 0.7566
Epoch 8/30 1563/1563 [==============================] - 57s 37ms/step - loss: 0.7182 - acc: 0.7574 - val_loss: 0.7874 - val_acc: 0.7540
Epoch 9/30 1563/1563 [==============================] - 60s 38ms/step - loss: 0.7018 - acc: 0.7615 - val_loss: 0.8130 - val_acc: 0.7416
Epoch 10/30 1563/1563 [==============================] - 59s 38ms/step - loss: 0.6753 - acc: 0.7720 - val_loss: 0.6432 - val_acc: 0.7921
Epoch 11/30 1563/1563 [==============================] - 59s 38ms/step - loss: 0.6558 - acc: 0.7787 - val_loss: 0.7087 - val_acc: 0.7828
Epoch 12/30 1563/1563 [==============================] - 60s 38ms/step - loss: 0.6467 - acc: 0.7818 - val_loss: 0.6456 - val_acc: 0.8043
Epoch 13/30 1563/1563 [==============================] - 59s 38ms/step - loss: 0.6330 - acc: 0.7885 - val_loss: 0.7529 - val_acc: 0.7810
Epoch 14/30 1563/1563 [==============================] - 59s 38ms/step - loss: 0.6198 - acc: 0.7919 - val_loss: 0.6237 - val_acc: 0.7995
Epoch 15/30 1563/1563 [==============================] - 59s 38ms/step - loss: 0.6154 - acc: 0.7959 - val_loss: 0.5923 - val_acc: 0.8106
Epoch 16/30 1563/1563 [==============================] - 59s 38ms/step - loss: 0.6038 - acc: 0.7996 - val_loss: 0.5909 - val_acc: 0.8108
Epoch 17/30 1563/1563 [==============================] - 59s 38ms/step - loss: 0.5893 - acc: 0.8041 - val_loss: 0.6112 - val_acc: 0.8089
Epoch 18/30 1563/1563 [==============================] - 59s 38ms/step - loss: 0.5841 - acc: 0.8055 - val_loss: 0.6016 - val_acc: 0.8132
Epoch 19/30 1563/1563 [==============================] - 59s 38ms/step - loss: 0.5758 - acc: 0.8100 - val_loss: 0.5788 - val_acc: 0.8219
Epoch 20/30 1563/1563 [==============================] - 58s 37ms/step - loss: 0.5647 - acc: 0.8130 - val_loss: 0.5923 - val_acc: 0.8100
Epoch 21/30 1563/1563 [==============================] - 58s 37ms/step - loss: 0.5543 - acc: 0.8159 - val_loss: 0.5328 - val_acc: 0.8305
Epoch 22/30 1563/1563 [==============================] - 58s 37ms/step - loss: 0.5544 - acc: 0.8179 - val_loss: 0.6232 - val_acc: 0.8074
Epoch 23/30 1563/1563 [==============================] - 59s 37ms/step - loss: 0.5448 - acc: 0.8192 - val_loss: 0.5510 - val_acc: 0.8264
Epoch 24/30 1563/1563 [==============================] - 58s 37ms/step - loss: 0.5405 - acc: 0.8214 - val_loss: 0.5377 - val_acc: 0.8300
Epoch 25/30 1563/1563 [==============================] - 59s 38ms/step - loss: 0.5343 - acc: 0.8233 - val_loss: 0.5028 - val_acc: 0.8392
Epoch 26/30 1563/1563 [==============================] - 58s 37ms/step - loss: 0.5234 - acc: 0.8260 - val_loss: 0.5112 - val_acc: 0.8397
Epoch 27/30 1563/1563 [==============================] - 59s 38ms/step - loss: 0.5233 - acc: 0.8259 - val_loss: 0.6078 - val_acc: 0.8141
Epoch 28/30 1563/1563 [==============================] - 58s 37ms/step - loss: 0.5206 - acc: 0.8290 - val_loss: 0.5830 - val_acc: 0.8278
Epoch 29/30 1563/1563 [==============================] - 59s 38ms/step - loss: 0.5054 - acc: 0.8317 - val_loss: 0.5649 - val_acc: 0.8241
Epoch 30/30 1563/1563 [==============================] - 59s 38ms/step - loss: 0.5092 - acc: 0.8311 - val_loss: 0.5535 - val_acc: 0.8323
10000/10000 [==============================] - 3s 289us/step Test loss: 0.553475624370575 Test accuracy: 0.8323
```

From <https://mq5xak7y2v-colab.googleusercontent.com/v2/usercontent/8b5e8f2bbe60490e/outputframe.html?vrz=colab-20181010-085300-RC00_216537743>