Lesson 5. Going Further with CNNs

5.1. Overfitting

- **= Model memorizes all the training set** so that the result on the test set is much lower than on the training set.

- **To avoid Overfitting,**

1. `Early Stopping` by using **validation set**.
    - We can see at which point overfitting is occurred by using validation set. We can ask couldn't we use the test set as a validation set? However, the problem is we ultimately end up tuning our models such that it performs well on both the training set and validation set.
    - Hence, we need separate test set to really see how our model generalizes and performs when given new data it has never seen before.
2. **Get more data** or do `Augmentation`
    - Image Augmentation has an effect that experiences model more different images to generalize the results via image transformation techniques.
3. `Dropout`

    `tf.keras.layers.Dropout()`

    - Randomly turn off some neurons not to allow certain neurons to get too much weighs
4. `Noise`
    - It seems it has no difference with data augmentation, but it can make model more robust to natural perturbations it could encounter in the wild.

5. `L1 & L2 Regularization`
    - Adding a penalty to the loss function
6. `Simplify the model`
    - If, even with all the data you now have, your model *still* manages to overfit your training dataset, it may be that the model is **too powerful**. You could then try to **reduce the complexity** of the model.
    - As stated previously, a model can only overfit *that much* data. By progressively reducing its complexity—*# of* ***estimators*** *in a* ***random forest****, # of* ***parameters*** *in a* ***neural network*** *etc.*—you can make the model *simple* enough that it *doesn't overfit,* but *complex* enough to *learn* from your data. To do that, it's convenient to look at the **error** on **both datasets** depending on the model complexity.
    - This also has the advantage of making the model **lighter**, **train faster** and **run faster**.

5.2. ImageDataGenerator

- **Import**

```
# ImageDataGenerator Options:
# rescale, horizontal_flip, rotaion_range,
# width_shift, height_shift_range, zoom_range_range,
# brightness_range, fill_mode and so on..
# + validation_split
image_gen = ImageDataGenerator(
                 rescale=1/255,
                 horizontal_flip=True,
                 rotation_range=45,
                 zoom_range=0.5,
                 width_shift_range=0.15,
                 height_shift_ragne=0.15)

# .flow_from_directory Options:
# batch_size, directory, shuffle, target_size,
# color_mode, class_mode, seed and so on...
train_data_gen =
image_gen.flow_from_directory(
        batch_size=batch_size,
        directory=train_dir_path,
        shuffle=True,
        target_size=(IMG_SHAPE, IMG_SHAPE))

train_data_gen[0] # a step of epoch. A tuple of images a
train_data_gen[0][0] # images of a whole batch.
train_data_gen[0][1] # labels of a whole batch
train_data_gen[0][0][0] # an image item.
```

```
# training!
EPOCHS = 100
# training
# actually, model.fit also works.
history = model.fit_generator(
    train_data_gen,
    steps_per_epoch=int(
        np.ceil(train_data_gen.n / float(BATCH_SIZE))),
    epochs=EPOCHS,
    validation_data=val_data_gen,
    validation_steps=int(
        np.ceil(val_data_gen.n / float(BATCH_SIZE)))
)
```