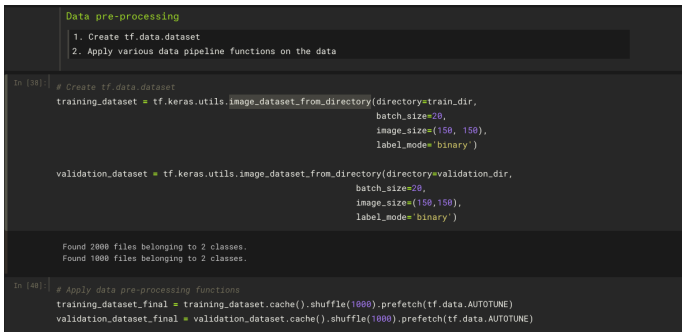
Course 2 - Convolutional Neural Networks using TensorFlow

Week 1

Minimal data pre-processing steps

1. Create a tensorflow dataset using the `image_dataset_from_directory` function
2. Use some functions from the data pipeline provided by tensorflow
   a. cache( )
   b. shuffle(buffer_size)
   c. prefetch(type) - A good possible value of the type is `tf.data.AUTOTUNE`

Definitions are provided in the first course's notes



New technique for `EarlyStopping`

- Directly import the `EarlyStopping` class from callbacks
- Monitor - the `validation loss`
- Patience - Number of epochs without any improvement after which training is stopped
- restore best weights - Restores the weights from the best epoch
- verbose - 1 displays why the callback stopped the execution

```
from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss',
                               patience=3,
                               restore_best_weights=True,
                               verbose=1)
```

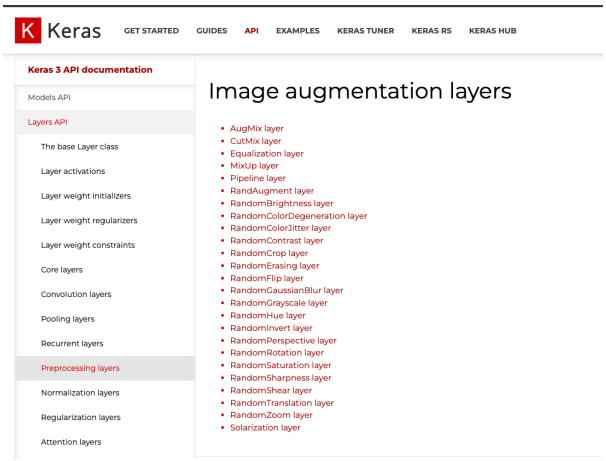
Week 2

Image Augmentation

Image Augmentation is a very simple, but very powerful tool to help you avoid overfitting your data. The concept is very simple though: If you have limited data, then the chances of you having data to match potential future predictions is also limited, and logically, the less data you have, the less chance you have of getting accurate predictions for data that your model hasn't yet seen. To put it simply, if you are training a model to spot cats, and your model has never seen what a cat looks like when lying down, it might not recognize that in the future.

Augmentation simply amends your images on-the-fly while training using transforms like rotation. So, it could 'simulate' an image of a cat lying down by rotating a 'standing' cat by 90 degrees. As such you get a cheap way of extending your dataset beyond what you have already.

https://keras.io/api/layers/preprocessing_layers/image



Some key points

- Sometimes even with image augmentation, there is a possibility that the model won't perform properly on the dataset
- This could be because the images are too diverse or if the validation dataset isn't designed properly

Week 3

Transfer Learning

The process of getting a pre-trained model locking its layers and then adding custom layers on the top of them

Example: *Consider getting a pre-trained model for people's faces, here the locked layers would be the*

Dropout (Regularization technique)

Mostly used to avoid Overfitting

The idea behind it is to remove a random number of neurons in your neural network. This works very well for two reasons: The first is that neighboring neurons often end up with similar weights, which can lead to

ones that have learned eyebrows, eyes, nose and all those features. Hence, our model doesn't need to learn all of them from scratch. We can now add final layers on top of these to finally create a face detection model.

overfitting, so dropping some out at random can remove this. The second is that often a neuron can over-weigh the input from a neuron in the previous layer, and can over specialize as a result. Thus, dropping out can break the neural network out of this potential bad habit!

A neuron cannot rely on any one input from previous layer because any of the previous layer neurons can get dropped at random.

More the parameters you can dropout more neurons so that overfitting doesn't occur.

Mostly used in *Computer Vision* because we have a lot of pixels

---

Feature of `tf.keras.utils.image_dataset_from_directory`

When we use this utility function, we don't specify the channel of image i.e. the 3 in (150,150,3) which represents the size of color channel.

This is because the utility assumes the channel automatically, which is set to 3 for images for instance *Cats vs Dogs dataset from Kaggle*. If we have grayscale images in the dataset, we would use the option `color_mode = 'grayscale'`

Methods to create models with TensorFlow and Keras

1. Using `Sequential API` - Using `tf.keras.Sequential`
2. Using `Functional API` - Using `tf.keras.Model` and using the output from previous layer as the input to the next layer
3. Subclassing the `Model` class