Data augmentation / aumento de dados é usado para aumentar o dataset, gerando variações que sejam treinadas

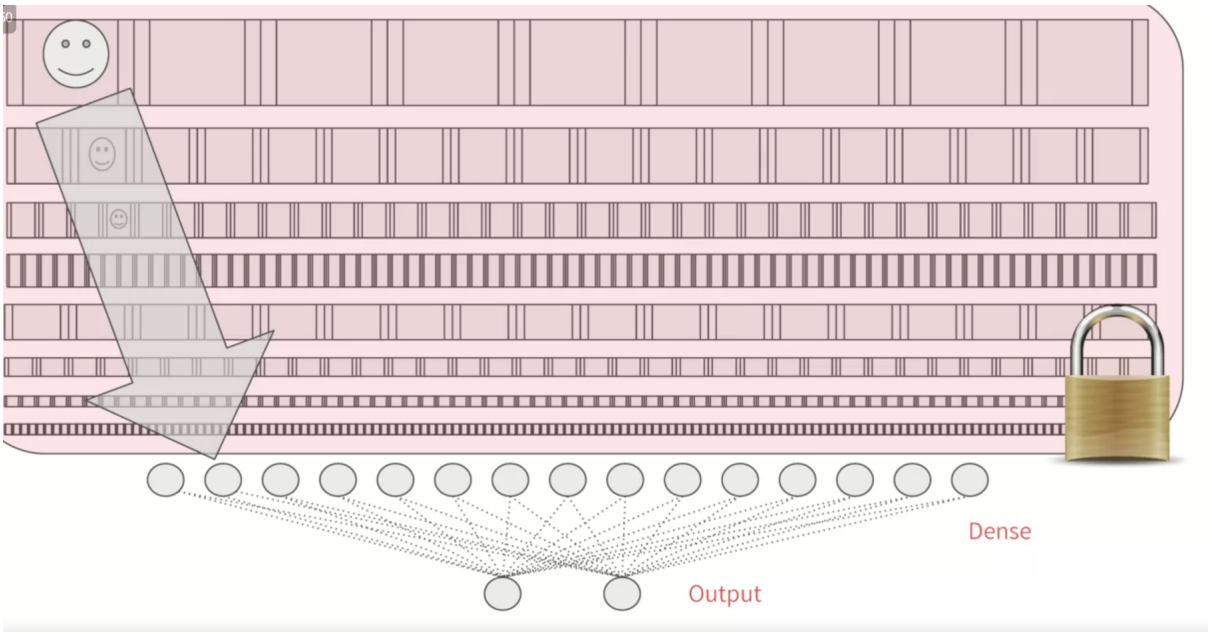
```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

O próprio ImageDataGenerator pode ter parâmetros que fazem algumas modificações, como mudar a rotação (de 0 a 40 degraus, no exemplo), shift muda a imagem dentro do frame, shearing faz algo assim, mudando o eixo, detectando mais facilmente pessoas deitadas, por exemplo:

fill mode mantém os pixels mais próximos (é pra não bugar muito a imagem)
Geralmente usando augmentation tem a tendência de os resultados do training set perderem a acurácia enquanto os resultados do validation set aumentarem a acurácia e ficarem mais próximos dos resultados do training set.
O imagegenerator faz um tudo em memória, não cria copias da original em disco, etc

Pode acontecer de ser inutil, pois mesmo com o data augmentation ainda não é possível representar as features desejadas para que os treinamentos melhorem.

Transfer Learning: Usar pesos de modelos ja treinados em nosso dataset

- Usar um modelo que já foi treinado em muitos mais dados e usar os features que esse modelo aprendeu para no final nosso modelo se apoiar nele e aprender a partir deste
- 

- Tem modelos bem famosos como o inception::
- ```
from tensorflow.keras.applications.inception_v3 import InceptionV3

local_weights_file = '/tmp/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'

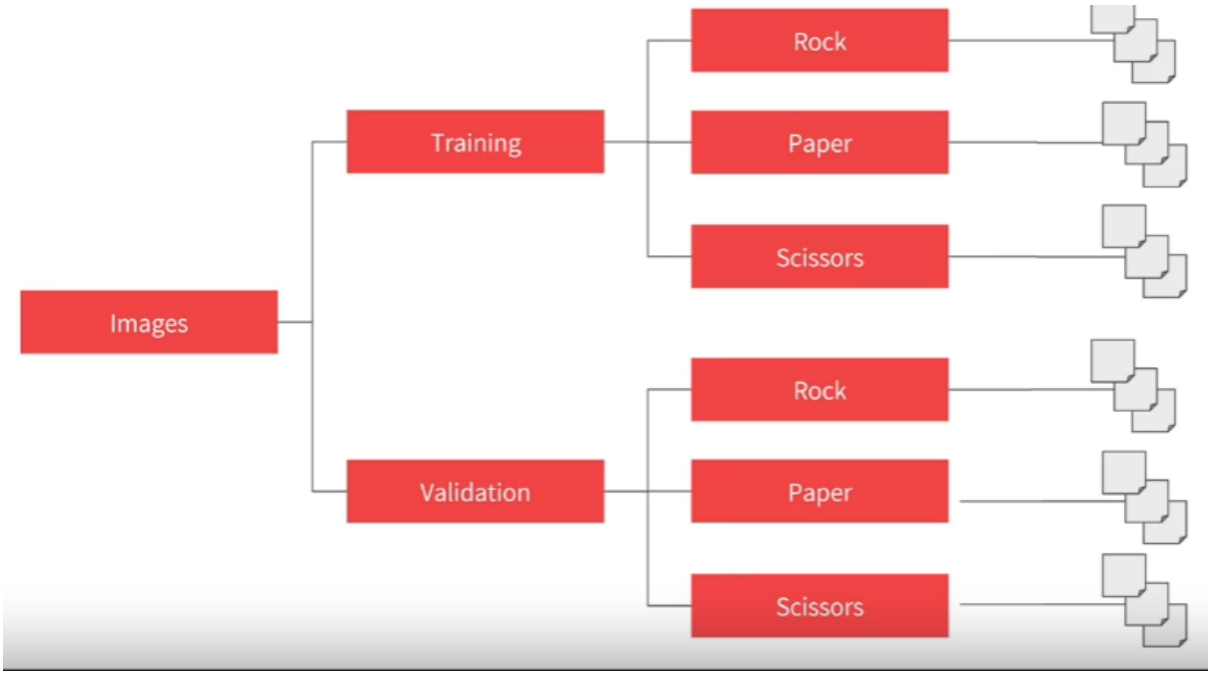
pre_trained_model = InceptionV3(input_shape = (150, 150, 3),
                                include_top = False,
                                weights = None)

pre_trained_model.load_weights(local_weights_file)
```
- Coloca o input shape das nossas imagens , em weights especificamos que não queremos usar os built in weights, mas sim os que foram baixados na local file a cima , include_top especifica que queremos ignorar a layer totalmente conectada no topo e ir diretamente para as convoluções do Inception
```

- Todas as layers tem nomes, podendo inspecionar ela:

```
last_layer = pre_trained_model.get_layer('mixed7')

last_output = last_layer.output
```

- **Dropout:** Previnir overfitting, remove um número randomico de neurons da NN. Funciona bem pois geralmente os neurons vizinhos tem pesos similares (gera overfitting) , e também pois um neuron pode geralmente considerar o peso de um input de um neuron maior que realmente é.
- **Lidar com multiclass classifiers** é diferente, no caso binário tinhamos duas classes em que as subpastas eram divididas, aqui temos várias pastas, uma para cada label possível
- 

- Algumas mudanças, mudar o class mode para categorical, mudar a ativação para algo que seja utilizável em multiclasses, como softmax que usa das probabilidades,também tem que mudar o loss para categorical
-

```
train_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(300, 300),
        batch_size=128,
        class_mode='categorical')
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu',
                           input_shape=(300, 300, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])
```

```
from tensorflow.keras.optimizers import RMSprop

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(lr=0.001),
              metrics=['acc'])
```