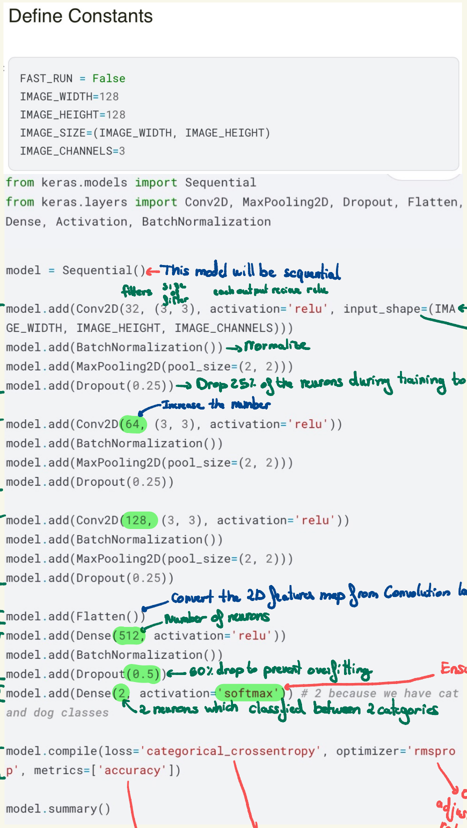
Sequential:
Input
First layer
Last layer
Output



Define Constants

```
FAST_RUN = False
IMAGE_WIDTH=128
IMAGE_HEIGHT=128
IMAGE_SIZE=(IMAGE_WIDTH, IMAGE_HEIGHT)
IMAGE_CHANNELS=3
```

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten,
Dense, Activation, BatchNormalization


model = Sequential()
```

← This model will be sequential

filters, size of filter, each output recive relu

First Convolutional Layer

```
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(IMA
GE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

← A 2D convolutional layer with 32 filters of size 3x3

→ Shape (width, heigth and channels)

↳ 3 for RGB

→ Normalize

→ Drop 25% of the neurons during training to prevent overfitting

Second Convolutional Layer

Increase the number

```
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

Third convolutional layer

```
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

Convert the 2D features map from Convolution layers into a 1D vector to pass it to the dense Layers

Flattening →

```
model.add(Flatten())
```

Number of neurons

Fully connected layers →

```
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
```

← 50% drop to prevent overfitting

Output layer →

```
model.add(Dense(2, activation='softmax')) # 2 because we have cat
and dog classes
```

Ensures that output is a probability distibution over 2 classes

↖ 2 neurons which classfied between 2 categories

Compiling the model

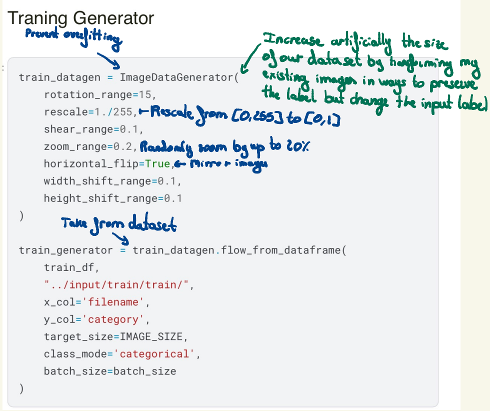
```
model.compile(loss='categorical_crossentropy', optimizer='rmspro
p', metrics=['accuracy'])

model.summary()
```

Accuracy will be the evaluation metric during training and testing

It's a multiclass classification model

Optimiser that adjust the learning rate during training



Traning Generator

Prevent overfitting

Increase artificially the size of our dataset by tranforming my existing images in ways to preseve the label but change the input label

→ Rotation
→ flipping
→ zooming

```
train_datagen = ImageDataGenerator(
    rotation_range=15,
    rescale=1./255,
    shear_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1
)
```

← Rescale from [0,255] to [0,1]

Randomly zoom by up to 20%

← Mirror images

Take from dataset

```
train_generator = train_datagen.flow_from_dataframe(
    train_df,
    "../input/train/train/",
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical',
    batch_size=batch_size
)
```