# CNN input Size explained

Most Imagenet pretrained CNNs were trained on 224x224 image resolution. It is a common misconception, that when using these pretrained CNN, images need to be resized to 224x224. On the contrary, popular CNN are fully convolutional nets that can accept any input size.

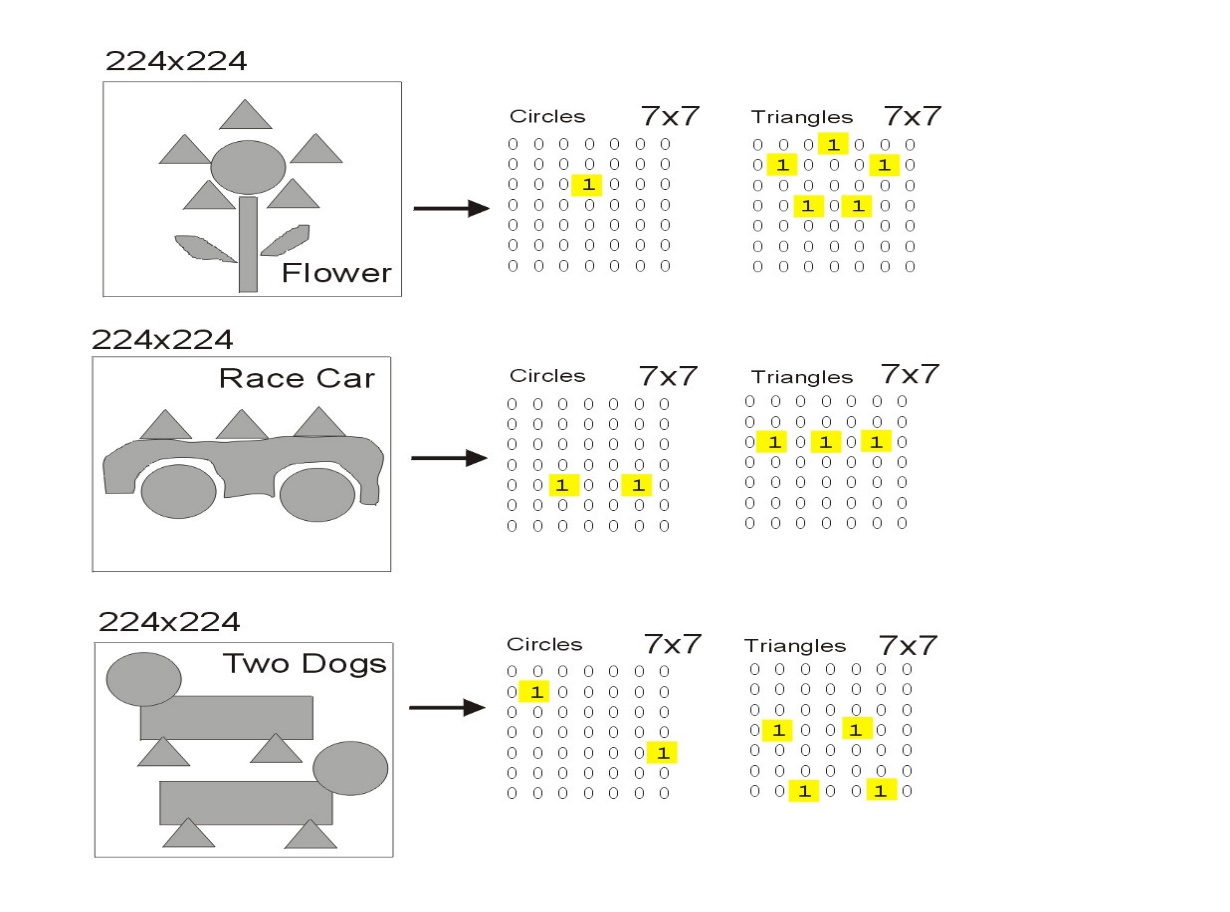
You can input any image size and these CNN output feature maps that are 32x times smaller. For example, if you input 224x224 then the CNN outputs feature maps of size 7x7. If you input images of size 512x512, then these CNN outputs feature maps of size 16x16.

# **Feature Maps**

The only relevance of the pretraining 224x224 size is that these CNN have learned to find certain patterns of **certain sizes**. For example, maybe they learned to find circles that are 50 pixels diameter, or maybe they learned to find triangles with side length 30 pixels.

**Pretrained Imagenet CNN**

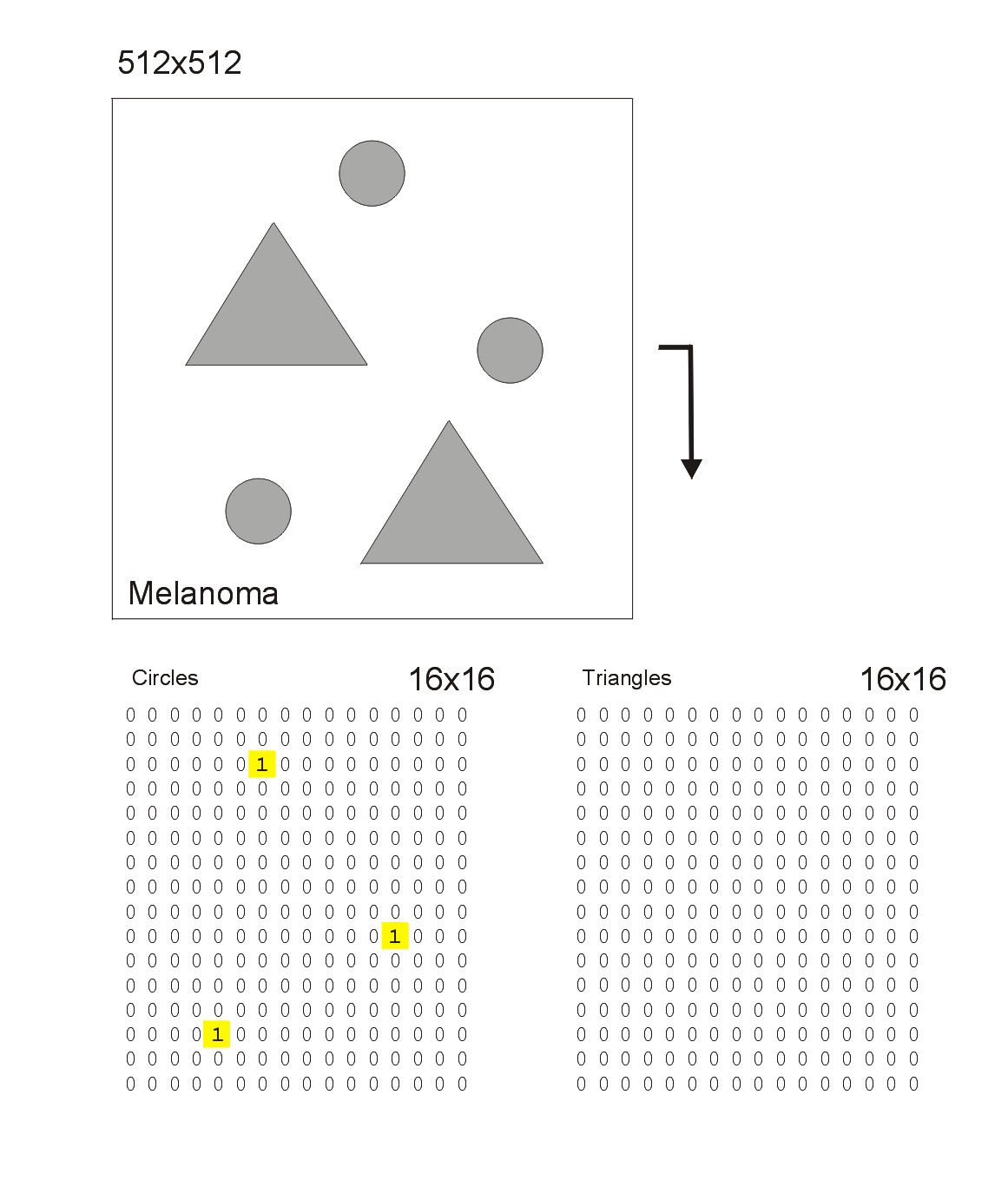
In the example below, we pretrain CNN on images of size 224x224 and they learn to detect circles of diameter 50 pixels and triangles of side length 30.



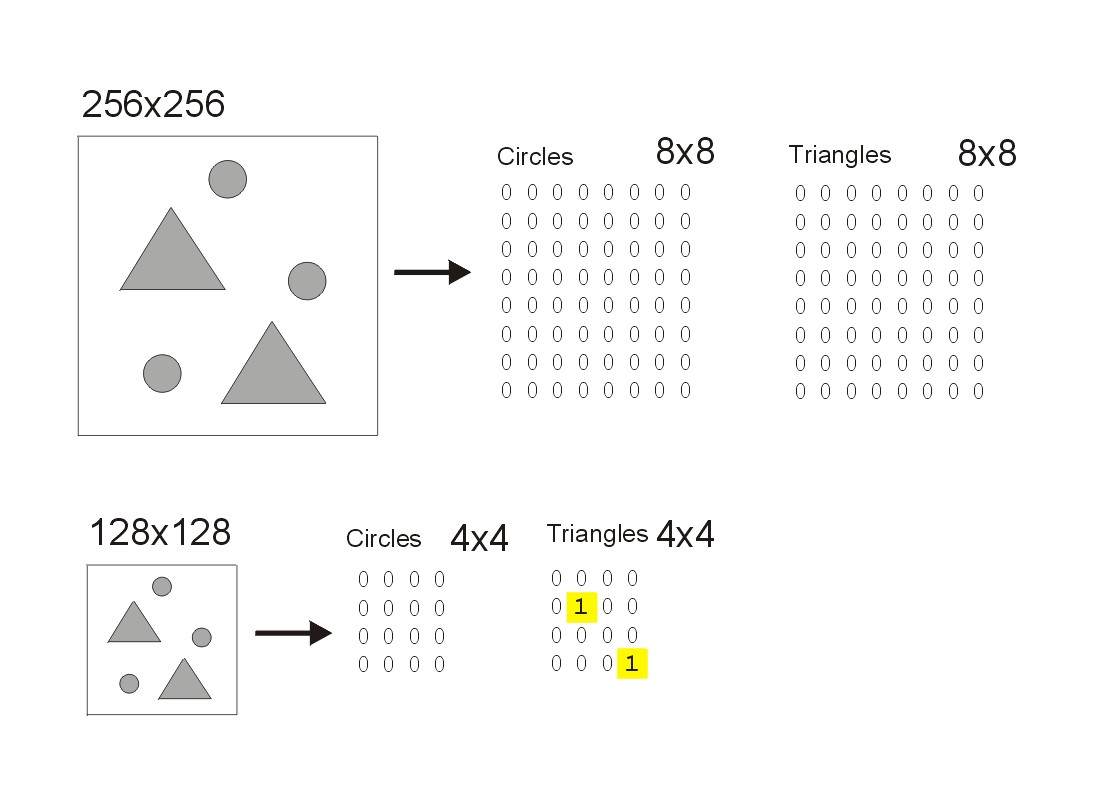
# **Why Resize Input Images**

When you resize input images, you change the size of your circles and triangles. So depending on how you resize your input image, this pretrained CNN may or may not find circles of diameter 50 and triangles of side 30. In the example below, given the original image, the CNN only finds the circles when the input image is resized to 512x512 and finds triangles when resized to 128x128

# **512x512 Input**



# **256x256 and 128x128 Input**



# **Conclusion**

In conclusion, a CNN searches for thousands of patterns (not just one circle size and one triangle size). It will find some patterns with some sizes and other patterns with other sizes. Therefore you should experiment with different input sizes to see which has better CV and LB. Also consider ensembling models of different input sizes for maximum CV and LB.

# **Kaggle Datasets**

I have resized all the competition images to TFRecords 768x768 [here](https://www.kaggle.com/cdeotte/melanoma-768x768), 512x512 [here](https://www.kaggle.com/cdeotte/melanoma-512x512), 384x384 [here](https://www.kaggle.com/cdeotte/melanoma-384x384), 256x256 [here](https://www.kaggle.com/cdeotte/melanoma-256x256), 192x192 [here](https://www.kaggle.com/cdeotte/melanoma-192x192), and I provide external data of size 512x512 [here](https://www.kaggle.com/cdeotte/512x512-melanoma-tfrecords-70k-images). If you ensemble models of different sizes, you can achieve LB 0.960 or higher!

If you prefer JPEGs instead of the TFRecords, you can find JPEGs 768x768 [here](https://www.kaggle.com/cdeotte/jpeg-melanoma-768x768), 512x512 [here](https://www.kaggle.com/cdeotte/jpeg-melanoma-512x512), 384x384 [here](https://www.kaggle.com/cdeotte/jpeg-melanoma-384x384), 256x256 [here](https://www.kaggle.com/cdeotte/jpeg-melanoma-256x256), 192x192 [here](https://www.kaggle.com/cdeotte/jpeg-melanoma-192x192). And external data 512x512 [here](https://www.kaggle.com/shonenkov/melanoma-merged-external-data-512x512-jpeg). (Also I have resized last year's 2019 data [here](https://www.kaggle.com/c/siim-isic-melanoma-classification/discussion/164910)).

You can even experiment with a single model that analyzes multiple image sizes at once with multiple EfficientNet backbones. After the image is inputted, you apply multiple types of downsizing like 2x, 3x, 4x to feed the different backbones. Then apply GlobalAveragePooling2D() after all the backbones then Concatenate all those vectors and finally apply Dense(1, activation='sigmoid') for classification. This worked well in Cloud Comp. An example is posted in Krazy Klassifiers [here](https://www.kaggle.com/c/understanding_cloud_organization/discussion/118086)

Enjoy!