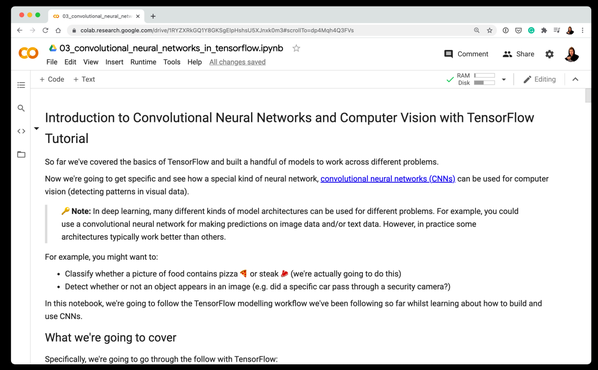
# Convolutional Neural Networks & Computer Vision with

TensorFlow

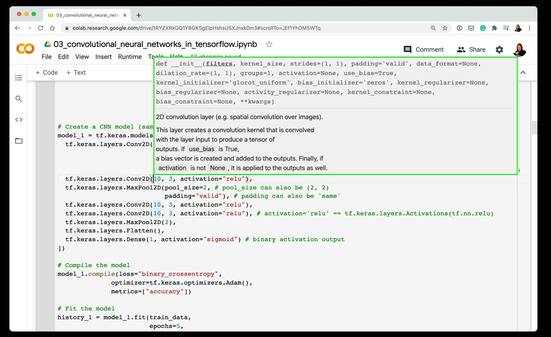
# Where can you get help?

- Follow along with the code




"If in doubt, run the code"

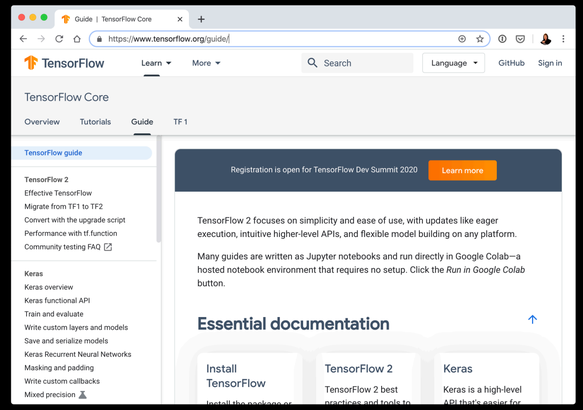
- Try it for yourself
- Press SHIFT + CMD + SPACE to read the docstring




- Search for it


- Try again
- Ask (don't forget the Discord chat!)

(yes, including the "dumb" questions)



“What is a computer vision problem?”

# Example computer vision problems

**Binary classification**

(one thing or another)

"Is this a photo of sushi, steak or pizza?"

**Multiclass classification**

(more than one thing or another)

**Object detection**

(where's the thing we're looking for?)

# What we're going to cover

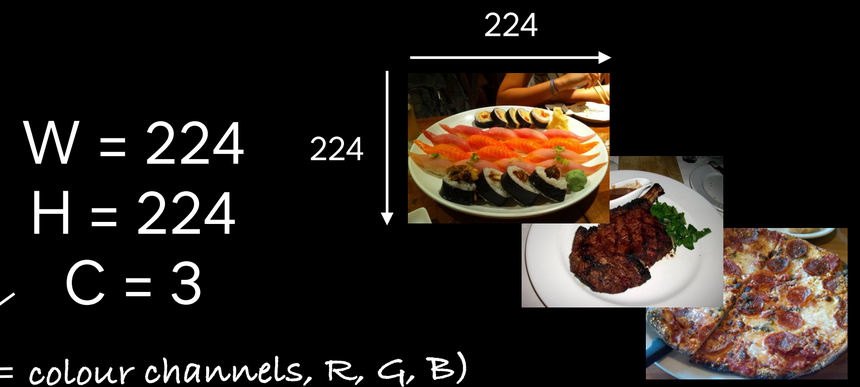
(broadly)

- Getting a dataset to work with (pizza_steak 🍕🥩)
- Architecture of a convolutional neural network (CNN) with TensorFlow
- An end-to-end binary image classification problem
- Steps in modelling with CNNs
  - Creating a CNN, compiling a model, fitting a model, evaluating a model
- An end-to-end multi-class image classification problem
- Making predictions on our own custom images

(we'll be cooking up lots of code!)

**How:**

# Computer vision inputs and outputs




W = 224
H = 224
C = 3

(c = colour channels, R, G, B)

Sushi
Steak
Pizza

**Actual output**

This is often a convolutional neural network (CNN)!

```
[[0.31, 0.62, 0.44…],
[0.92, 0.03, 0.27…],
[0.25, 0.78, 0.07…],
…,
```

(normalized pixel values)

**Numerical encoding**

Inputs

Machine Learning Algorithm

(often already exists, if not, you can build one)

Outputs

```
[[0.97, 0.00, 0.03],
[0.81, 0.14, 0.05],
[0.03, 0.07, 0.90],
…,
```

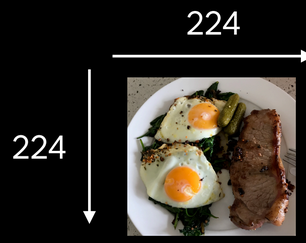
**Predicted output**

(comes from looking at lots of these)

# Input and output shapes

(for an image classification example)

We're going to be building CNNs to do this part!

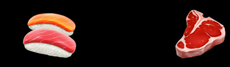



```
[[0.31, 0.62, 0.44…],
[0.92, 0.03, 0.27…],
[0.25, 0.78, 0.07…],
…,
```

Inputs

Machine Learning Algorithm

Outputs



```
[0.00, 0.97, 0.03]
```

(prediction probabilities)

(gets represented as a tensor)

```
[batch_size, width, height, colour_channels]
```

```
Shape = [None, 224, 224, 3]
```

or

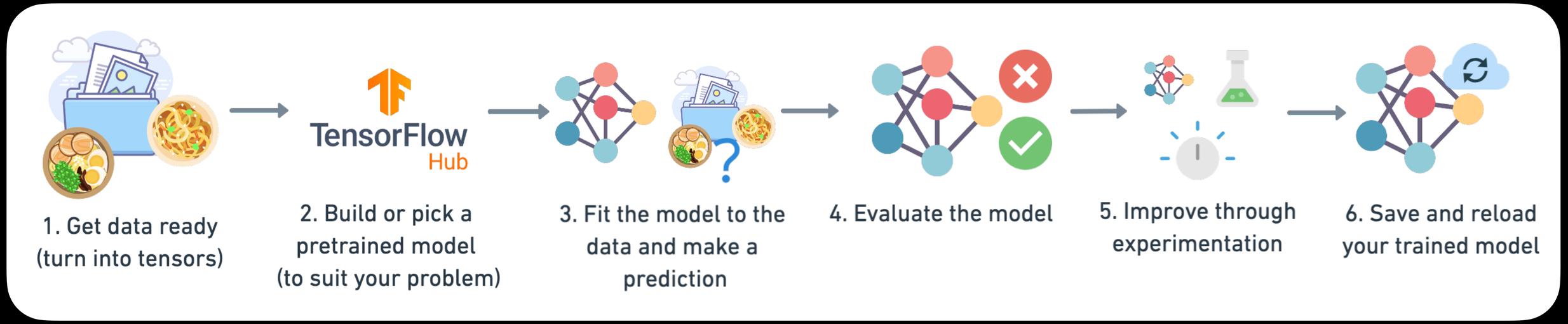
```
Shape = [32, 224, 224, 3]
```

(32 is a very common batch size)

```
Shape = [3]
```

These will vary depending on the problem you're working on.

# Steps in modelling with TensorFlow




1. Turn all data into numbers (neural networks can't handle images)
2. Make sure all of your tensors are the right shape
3. Scale features (normalize or standardize, neural networks tend to prefer normalization)

“What is a convolutional neural network (CNN)?”

(typical)*

# Architecture of a CNN

| Hyperparameter/Layer type | What does it do? | Typical values |
|---|---|---|
| Input image(s) | Target images you'd like to discover patterns in | Whatever you can take a photo (or video) of |
| Input layer | Takes in target images and preprocesses them for further layers | `input_shape = [batch_size, image_height, image_width, color_channels]` |
| Convolution layer | Extracts/learns the most important features from target imagaes | Multiple, can create with `tf.keras.layers.ConvXD` (X can be multiple values) |
| Hidden activation | Adds non-linearity to learned features (non-straight lines) | Usually ReLU ( `tf.keras.activations.relu` ) |
| Pooling layer | Reduces the dimensionality of learned image features | Average ( `tf.keras.layers.AvgPool2D` ) or Max ( `tf.keras.layers.MaxPool2D` ) |
| Fully connected layer | Further refines learned features from convolution layers | `tf.keras.layers.Dense` |
| Output layer | Takes learned features and outputs them in shape of target labels | `output_shape = [number_of_classes]` (e.g. 3 for pizza, steak or sushi) |
| Output activation | Adds non-linearities to output layer | `tf.keras.activations.sigmoid` (binary classification) or `tf.keras.activations.softmax` |

(what we're working towards building)

```python
# 1. Create a CNN model (same as Tiny VGG - https://poloclub.github.io/cnn-explainer/)
cnn_model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(filters=10,
                         kernel_size=3, # can also be (3, 3)
                         activation="relu",
                         input_shape=(224, 224, 3)), # specify input shape (height, width, colour channels)
  tf.keras.layers.Conv2D(10, 3, activation="relu"),
  tf.keras.layers.MaxPool2D(pool_size=2, # pool_size can also be (2, 2)
                            padding="valid"), # padding can also be 'same'
  tf.keras.layers.Conv2D(10, 3, activation="relu"),
  tf.keras.layers.Conv2D(10, 3, activation="relu"), # activation='relu' == tf.keras.layers.Activations(tf.nn.relu)
  tf.keras.layers.MaxPool2D(2),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(1, activation="sigmoid") # binary activation output
])

# 2. Compile the model
cnn_model.compile(loss="binary_crossentropy",
                  optimizer=tf.keras.optimizers.Adam(),
                  metrics=["accuracy"])

# 3. Fit the model
history = cnn_model.fit(train_data, epochs=5)
```

→ Steak 🥩 Pizza 🍕

*__Note:__ there are almost an unlimited amount of ways you could stack together a convolutional neural network, this slide demonstrates only one.

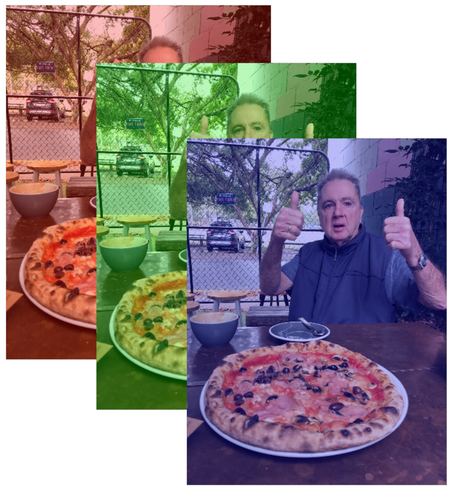
# Let’s code!

# Architecture of a CNN

(coloured block edition)

## Simple CNN

Inputs



Processed inputs
(these get represented as a tensor)

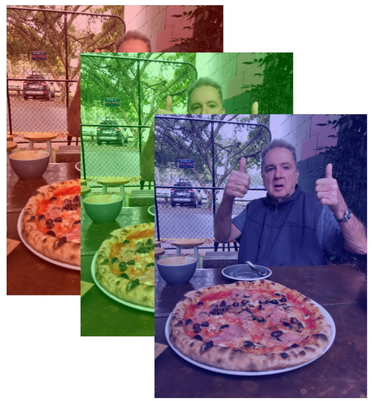
Convolutional layer
(+ ReLU activation)

Pooling
layer

These can be combined in many different ways

Fully connected
output layer

## Deeper CNN



Convolutional layer

Pooling
layer

Convolutional layer

Pooling
layer

Convolutional layer

Pooling
layer

# Breakdown of `Conv2D` layer

**Example code:** `tf.keras.layers.Conv2D(filters=10, kernel_size=(3, 3), strides=(1, 1), padding="same")`
**Example 2 (same as above):** `tf.keras.layers.Conv2D(filters=10, kernel_size=3, strides=1, padding="same")`

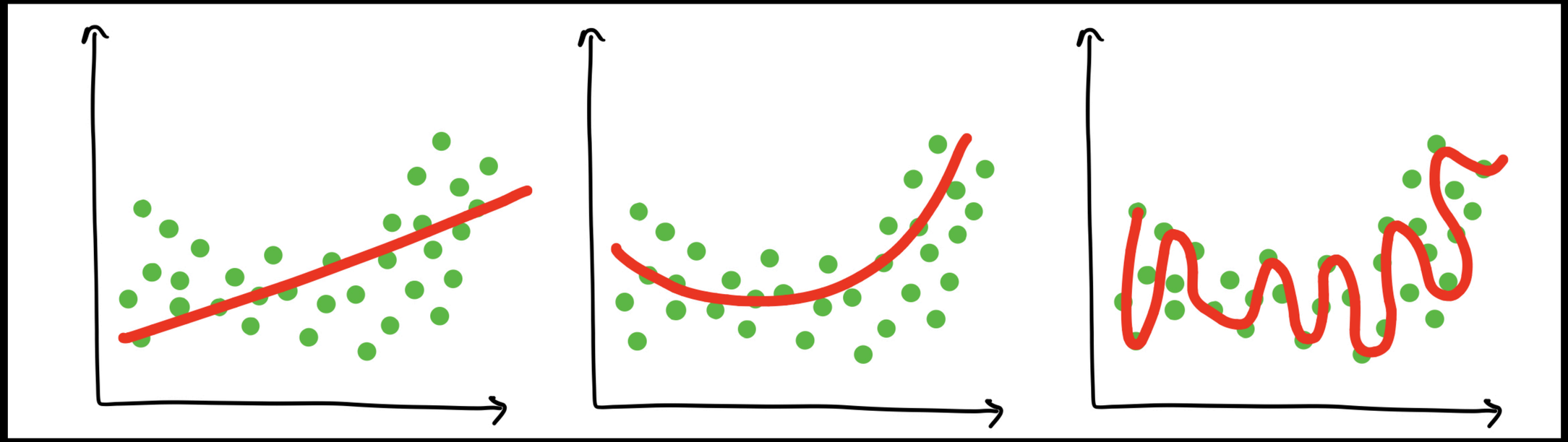
| Hyperparameter name | What does it do? | Typical values |
|---|---|---|
| Filters | Decides how many filters should pass over an input tensor (e.g. sliding windows over an image). | `10, 32, 64, 128 (higher values lead to more complex models)` |
| Kernel size (also called filter size) | Determines the shape of the filters (sliding windows) over the output. | `3, 5, 7 (lowers values learn smaller features, higher values learn larger features)` |
| Padding | Pads the target tensor with zeroes (if "`same`") to preserve input shape. Or leaves in the target tensor as is (if "`valid`"), lowering output shape. | "same" or "valid" |
| Strides | The number of steps a filter takes across an image at a time (e.g. if `strides=1`, a filter moves across an image 1 pixel at a time). | `1 (default), 2` |

📖 **Resource:** For an interactive demonstration of the above hyperparameters, see the CNN explainer website.

# What is overfitting?

**Overfitting** — when a model over learns patterns in a particular dataset and isn't able to generalise to unseen data.

For example, a student who studies the course materials too hard and then isn't able to perform well on the final exam. Or tries to put their knowledge into practice at the workplace and finds what they learned has nothing to do with the real world.



**Underfitting** | **Balanced** (goldilocks zone) | **Overfitting**

# Improving a model

(from a model's perspective)

```python
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(4, activation="relu"),
  tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.001),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train_subset, y_train_subset, epochs=5)
```

Smaller model



```python
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.0001),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train_full, y_train_full, epochs=100)
```

Larger model

## Common ways to improve a deep model:

- Adding layers
- Increase the number of hidden units
- Change the activation functions
- Change the optimization function
- Change the learning rate
- Fitting on more data
- Fitting for longer

(because you can alter each of these, they're hyperparameters)

# Improving a model

(from a data perspective)

| Method to improve a model (reduce overfitting) | What does it do? |
| --- | --- |
| More data | Gives a model more of a chance to learn patterns between samples (e.g. if a model is performing poorly on images of pizza, show it more images of pizza). |
| Data augmentation | Increase the diversity of your training dataset without collecting more data (e.g. take your photos of pizza and randomly rotate them 30°). Increased diversity forces a model to learn more generalisation patterns. |
| Better data | Not all data samples are created equally. Removing poor samples from or adding better samples to your dataset can improve your model's performance. |
| Use transfer learning | Take a model's pre-learned patterns from one problem and tweak them to suit your own problem. For example, take a model trained on pictures of cars to recognise pictures of trucks. |

# What is data augmentation?

Looking at the same image but from different perspective(s)*.

**Original**

**Rotate**

**Shift**

**Zoom**

***Note:** There are many more different kinds of data augmentation such as, cropping, replacing, shearing. This slide only demonstrates a few.

# Popular & useful computer vision architectures

| Architecture | Release Date | Paper | Use in TensorFlow | When to use |
|---|---|---|---|---|
| ResNet (residual networks) | 2015 | https://arxiv.org/abs/1512.03385 | Find pre-trained versions on TensorFlow Hub or `tf.keras.applications` | A good backbone for many computer vision problems |
| EfficientNet(s) | 2019 | https://arxiv.org/abs/1905.11946 | Find pre-trained versions on TensorFlow Hub or `tf.keras.applications` | Typically now better than ResNets for computer vision |
| MobileNet(s) | 2017 | https://arxiv.org/abs/1704.04861 | Find pre-trained versions on TensorFlow Hub or `tf.keras.applications` | Lightweight architecture suitable for devices with less computing power |

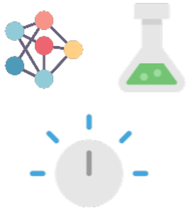
# Steps in modelling with TensorFlow

1. Get data ready (turn into tensors)


2. Build or pick a pretrained model (to suit your problem)

3. Fit the model to the data and make a prediction

4. Evaluate the model



5. Improve through experimentation

6. Save and reload your trained model

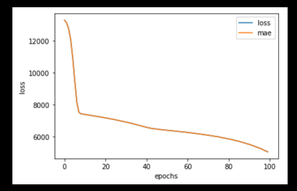
```python
# 1. Create a model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.Input(shape=(224, 224, 3)),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(3, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train, y_train, epochs=5)

# 4. Evaluate the model
model.evaluate(X_test, y_test)
```

# The machine learning explorer's motto

"Visualize, visualize, visualize"

**Data**

**Model**




**Training**

**Predictions**

It's a good idea to visualize these as often as possible.

# The machine learning practitioner's motto

"Experiment, experiment, experiment"


(try lots of things and see what tastes good)