
Quick, Draw!

Recognition of drawings with different model types.

Tobias Fischer & Stefan Penzinger

HAGENBERG | LINZ | STEYR | WELS



UNIVERSITY
OF APPLIED SCIENCES
UPPER AUSTRIA




[stefanpenzinger/ml-quick-draw](https://github.com/stefanpenzinger/ml-quick-draw)

Goal

- Recognize images based on Google's „Quick, draw!“ dataset
- Create tool to draw pictures in the necessary format
- Use different models
 - > K nearest neighbour (KNN)
 - > Random Forest (RF)
 - > Multilayer Perceptron (MLP)
 - > Convolutional Neural Network (CNN)
- Compare them on how they perform to self-drawn pictures

Dataset

- Google provides 50 million drawings across 345 categories
- Recorded in their game “Quick, Draw!”
 - > quickdraw.withgoogle.com
 -  [googlecreativelab/quickdraw-dataset](https://github.com/googlecreativelab/quickdraw-dataset)
- Raw Data
- Simplified Data (already pre-processed)
- Bitmap (simplified data converted to 28x28 greyscale bitmap)

requirements.txt

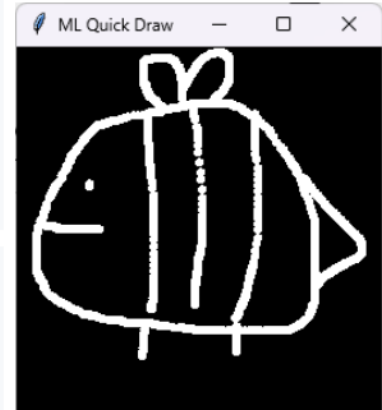
Python 3.10

- NumPy
- Pandas
- Matplotlib
- scikit-learn
- TensorFlow
 - > high level API for TensorFlow
 - > used for CNN
- ProtoBuf
 - > used in TensorFlow
- Pillow
 - > used for generating image from canvas

1st Approach – simplified vectors

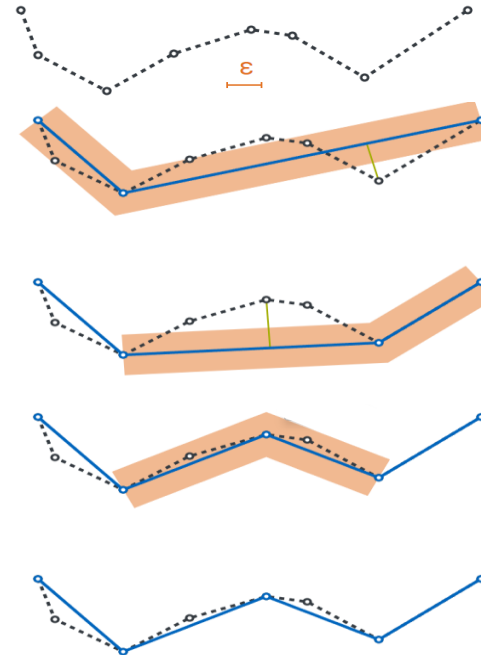
- Use .ndjson dataset from Google
- Use vectors to describe the paths of the image
 - > Top left corner is the origin (0, 0)
 - > Bottom right (255, 255)
 - > 1 pixel spacing of strokes
 - > Simplify strokes using Ramer-Douglas-Peucker (RDP) algorithm ($\epsilon = 2.0$)

```
General:
drawing: {
  [
    [ // First stroke
      [x0, x1, x2, x3, ...],
      [y0, y1, y2, y3, ...]
    ],
    [ // Second stroke
      [x0, x1, x2, x3, ...],
      [y0, y1, y2, y3, ...]
    ],
    ... // Additional strokes
  ]
}
Example of stroke for the bee's stinger:
drawing: {
  [ // stroke of bee's body ],
  [
    200, 242, 242, 238, 222, 209
  ],
  [
    91, 138, 145, 149, 156, 167
  ],
  // further strokes
}
```



RDP algorithm

- Used for smoothing polylines
 - > Done by reducing number of points
- Original shape should be preserved
- Parameter epsilon ϵ
 - > Defines max. distance between original points and simplified curve



(source: <https://cartography-playground.gitlab.io/playgrounds/douglas-peucker-algorithm/>)

Feature Extraction

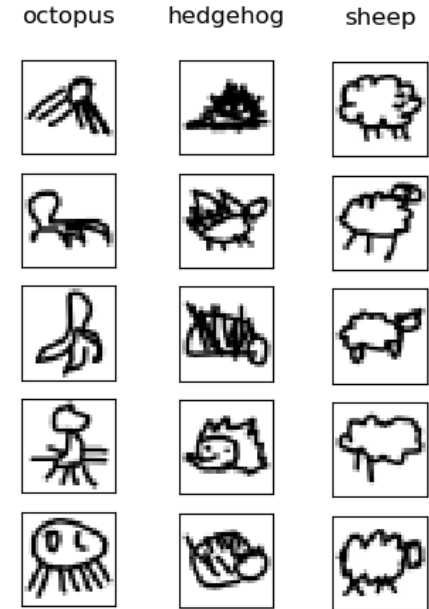
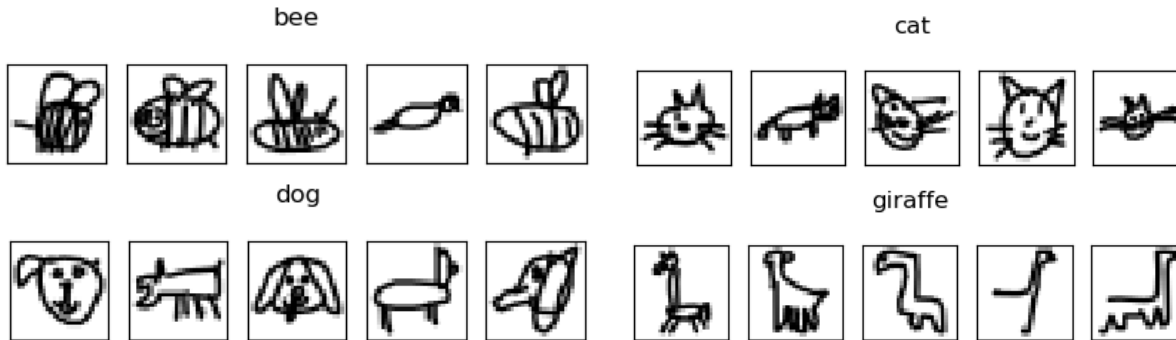
- Features:
 - > Datapoint Count: The total number of datapoints in the drawing.
 - > Stroke Count: The total number of strokes used to create the drawing.
 - > y-Max: The highest y-coordinate reached in the drawing.
 - > Datapoint Percentage of Stroke 0: The percentage of datapoints that belong to stroke 0.
 - > Datapoint Percentage of Stroke 1: The percentage of datapoints that belong to stroke 1.
 - > Datapoint Percentage of Stroke 2: The percentage of datapoints that belong to stroke 2.
 - > Direction Stroke 1: The direction of stroke 1.
 - > Direction Stroke 2: The direction of stroke 2.
 - > x0: The x-coordinate of the starting point of the drawing.

Model Training

- Train/test split of 75% used
- Model types: KNN, Random Forest and Decision Tree
- Tried out several hyperparameter variations per model
- Accuracy of model performed rather bad
- Possible reasons:
 - > Feature extraction with way too less features
 - > Insufficient hyperparameter tuning
 - > Preprocessing of data was not done well enough

2nd Approach – bitmaps

- Using 28x28 bitmaps instead of vectors
- Distinct between 7 animals
 - > bee, cat, dog, giraffe, hedgehog, octopus and sheep



Model Training

- Bitmap values as raw features
 - > Normalized: divided by 255 \rightarrow [0,1]
 - > Flattened into 784-dimensional vector (28x28)
 - > CNN: directly used raster image format
- Appended with dummy variable for each class label
- 50% train-test split
- All models except CNN
 - > 10-fold CV
 - > Hyperparameter Grid Search
- Different amount samples per animal class examined
 - > 2,000, 5,000, 10,000 and 15,000 (not higher \rightarrow plateau)
- Utilized multiple cores with `n_jobs=-1`

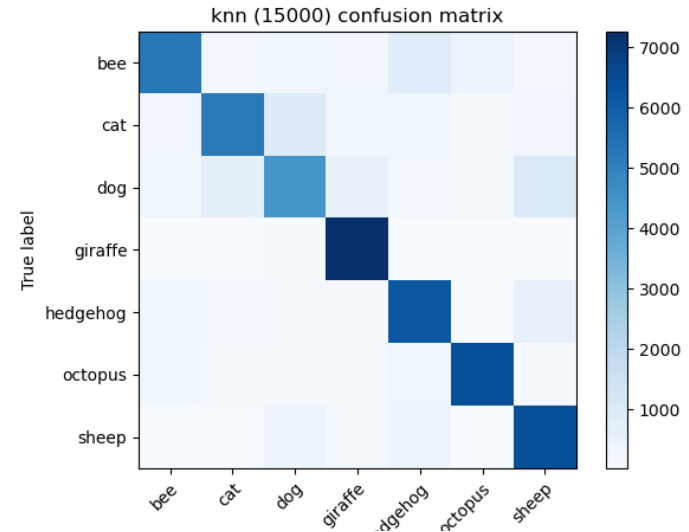
KNN

- Performed grid search
 - > $K = [1, 3, 5, 7, 9, 11]$
- Training results

Samples/Class	Best number of neighbours	Best training score
2,000	5	0.725
5,000	7	0.753
10,000	7	0.772
15,000	9	0.784

- Test Results

Samples/Class	Test Score
2,000	0.730
5,000	0.755
10,000	0.776
15,000	0.786



Random Forest

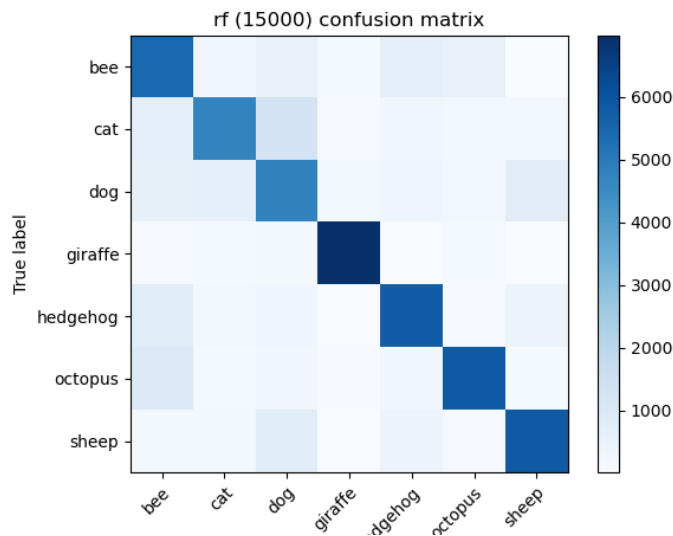
- Performed grid search
 - > `n_estimators = [10, 20, 40, 60, 80, 100, 120, 140, 160]`
 - > Results stagnated after 160

- Training results

Samples/Class	Best number of estimators	Best training score
2,000	160	0.704
5,000	160	0.722
10,000	160	0.740
15,000	160	0.748

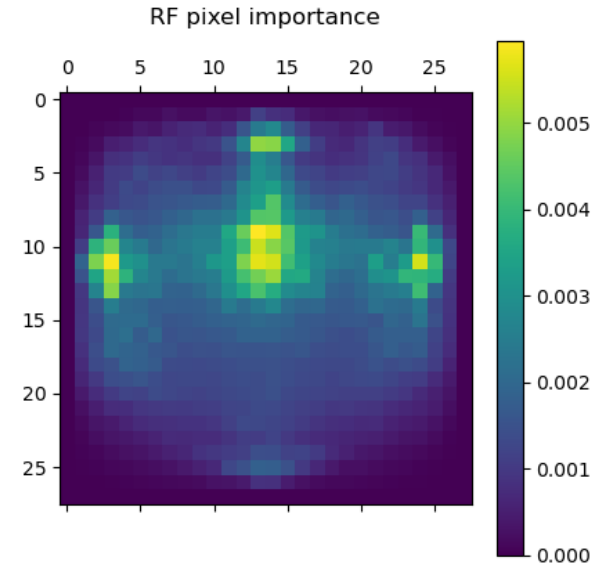
- Test results

Samples/Class	Test Score
2,000	0.712
5,000	0.728
10,000	0.741
15,000	0.750



RF – Pixel Importance

- Recap: Tree-based models
 - > More important features located near the root
- Our features: pixels
 - > Pixels around the corner are less important (dark blue/purple)
 - > Pixels e.g. in the middle are much more important (yellow)



MLP

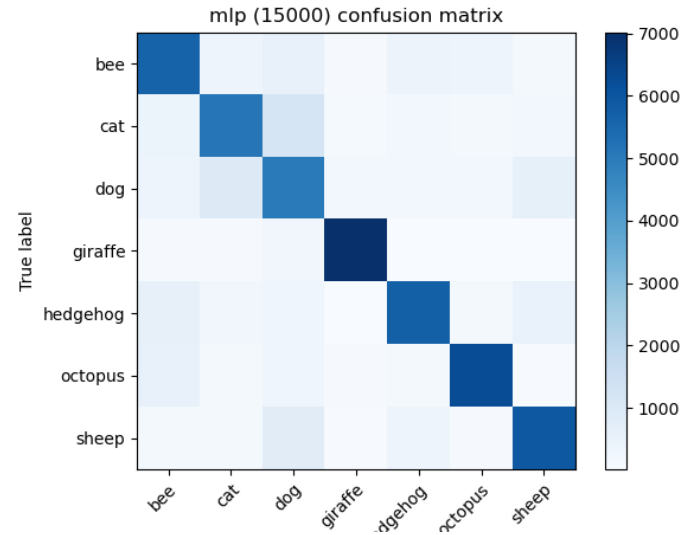
- Performed grid search
 - > Hidden Layer Sizes: (100), (100, 100)
 - > Alpha: $[10^{-4}, 10^{-1}]$

- Training Results

Samples/Class	Best hidden layer sizes	Alpha	Best training score
2,000	(100)	0.1	0.702
5,000	(100, 100)	0.01	0.734
10,000	(100, 100)	0.001	0.757
15,000	(100)	0.1	0.768

- Test Results

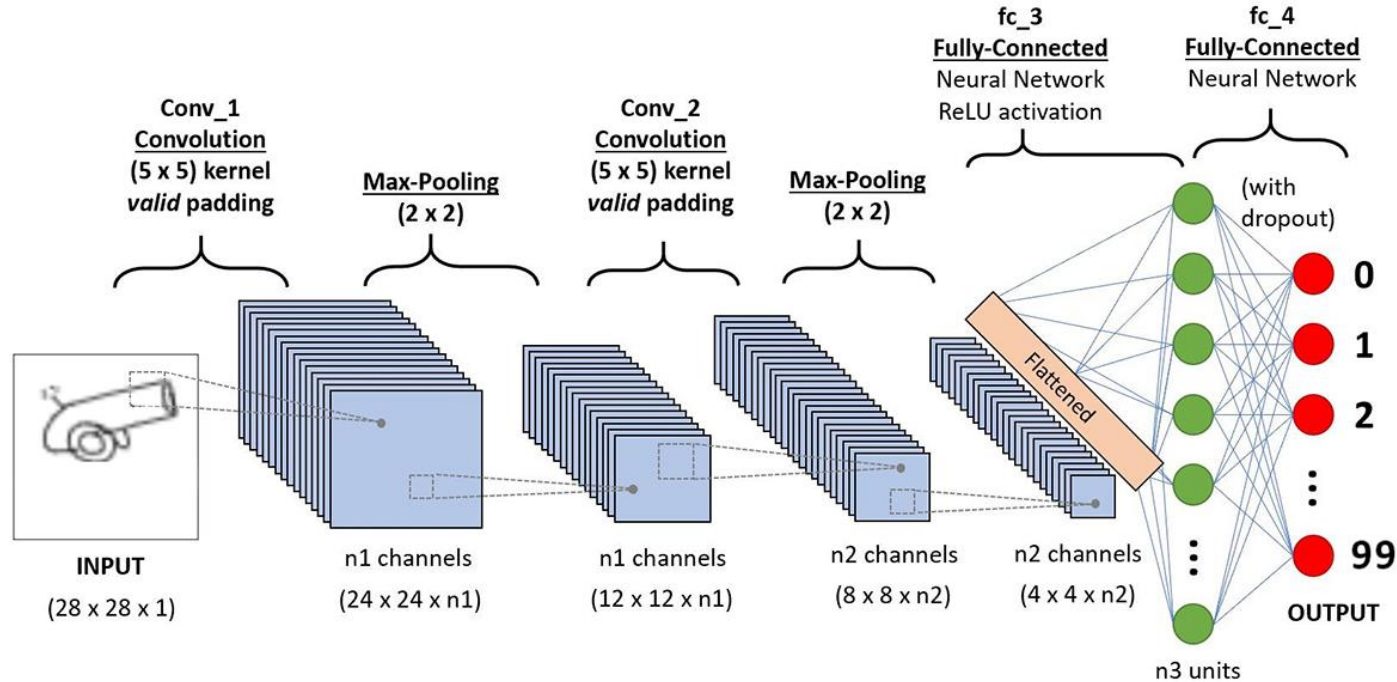
Samples/Class	Test Score
2,000	0.720
5,000	0.737
10,000	0.757
15,000	0.776



CNN (Convolutional Neural Network)

- Based on [1]
- Batch Size: 200
 - > Model processes 200 samples simultaneously
 - > After completing first batch, model continues training with next batch
 - > ...
- Training Epochs: 10
 - > Improvements stagnate

CNN – Layers [1]



CNN – Layers [1]

1. Convolutional layer with 30 feature maps of size 5×5
2. Pooling layer taking the max over 2×2 patches
3. Convolutional layer with 15 feature maps of size 3×3
4. Pooling layer taking the max over 2×2 patches
5. Flatten layer
6. Fully connected layer with 128 neurons and rectifier activation
7. Fully connected layer with 50 neurons and rectifier activation
8. Output layer

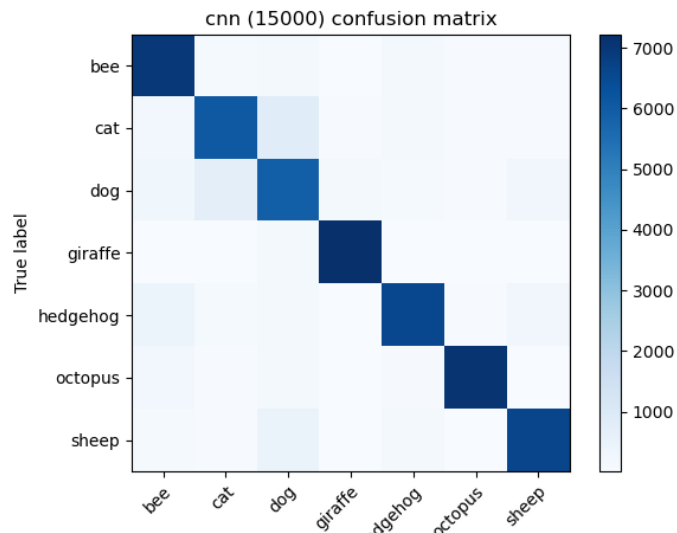
CNN

- Training/test results

Samples/Class (Time/Epoch)	Epoch 5 test accuracy	Epoch 8 test accuracy	Epoch 10 test accuracy
2,000 (1s/E)	0.751	0.791	0.804
5,000 (2s/E)	0.822	0.840	0.847
10,000 (5s/E)	0.850	0.870	0.874
15,000 (9s/E)	0.867	0.880	0.885

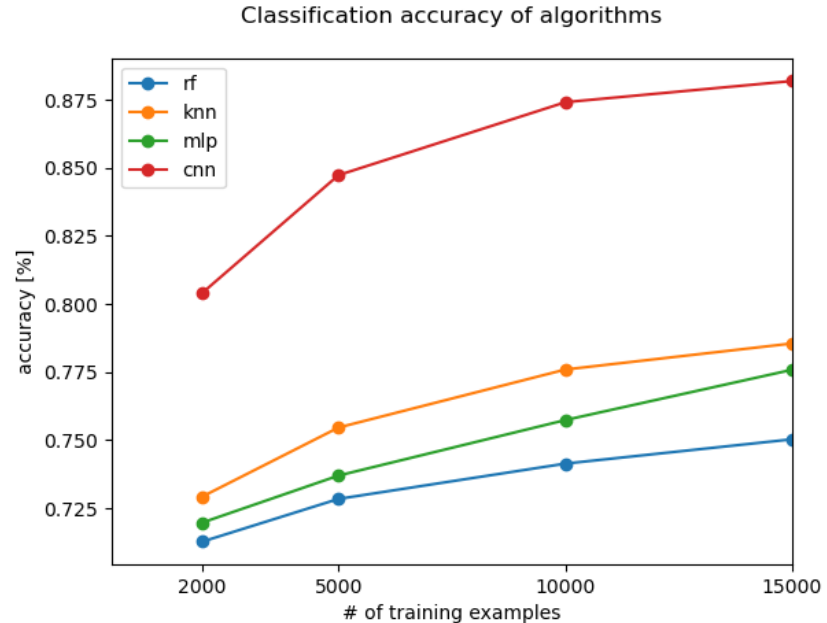
- Test results

Samples/Class	Test Score
2,000	0.804
5,000	0.847
10,000	0.874
15,000	0.882



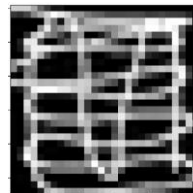
Model Evaluation

- Overall training duration: 2h
- Evaluate all models with held-back test
- Best performing model
 - > Per model type
 - > Output confusion matrix (as seen the slides before)
 - > Save for further usage
- Compare models by accuracy and sample size



Improvements

- Self-made Bitmap does not exactly contain the same information as Google's dataset
 - > exact scaling/centring of the pictures
 - > intensity of strokes (→ weight of intensity saved in bitmap)
- Better hyperparameter tuning
- Improved feature extraction
 - > E.g. remove unimportant areas (as seen in RF)
- Only 12.5% of overall samples used for training (15k of 120k)
- Filter out wild scribbled drawings
 - > Other data would be labelled as (not) recognized by Google
 - > Maybe transform data to bitmap by ourselves



Conclusion

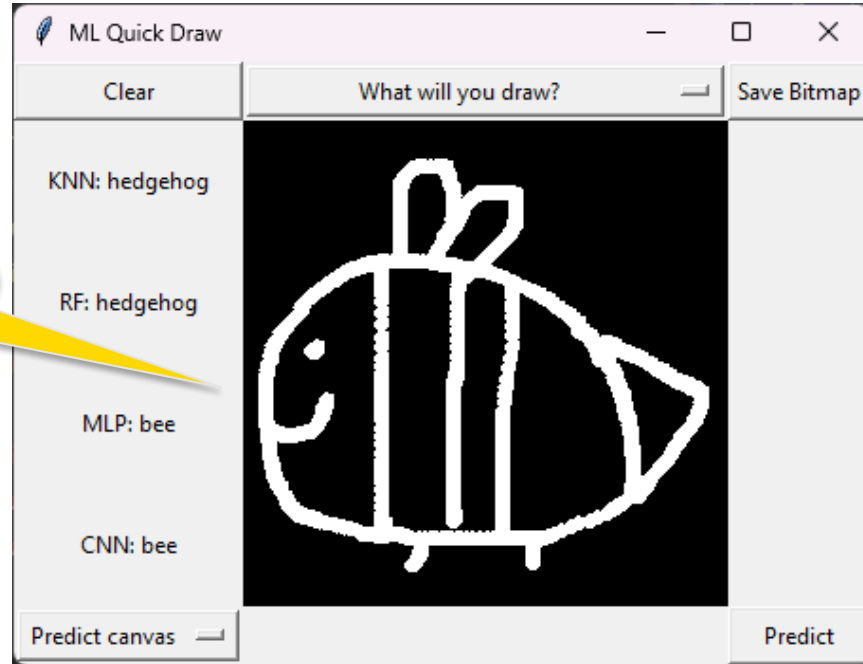
- CNN model efficient at recognizing drawings (Bitmap)
- Other models lack accuracy
 - > Our data would have to exactly fit the data of google
 - > **E.g. Random forest:**
many if-else statements which check if the pixel match any pixel in the trained data
- Drawings often misclassified as hedgehog
 - > Dataset often contains wild scribbles for hedgehogs
 - > Such as:



Live Demo

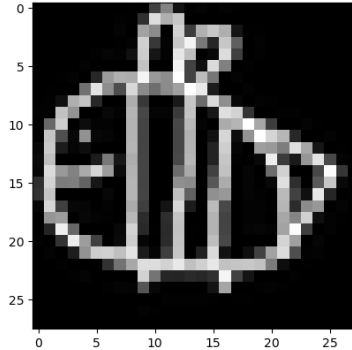
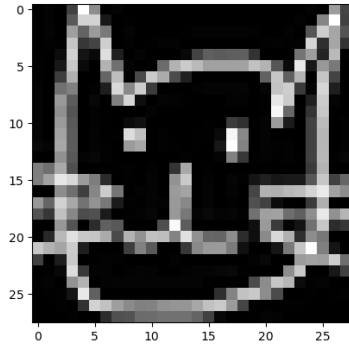


ICH BIN
EINE BIENE!

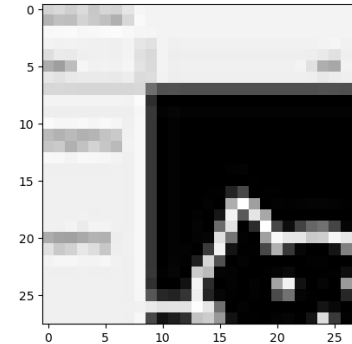


Live Demo

Should look like this (low pixel density):



Does look like this (on laptop screen):



Quick, Draw!

Recognition of drawings with different model types.

Tobias Fischer & Stefan Penzinger

HAGENBERG | LINZ | STEYR | WELS



UNIVERSITY
OF APPLIED SCIENCES
UPPER AUSTRIA



[stefanpenzinger/ml-quick-draw](https://github.com/stefanpenzinger/ml-quick-draw)