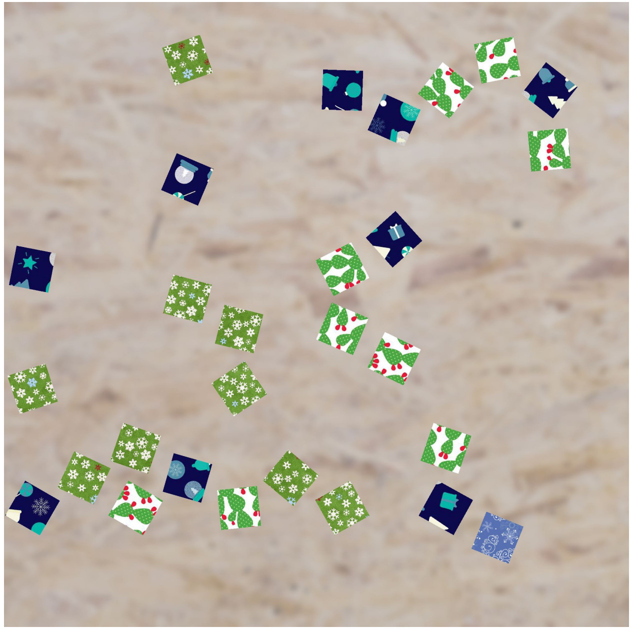
Solving tiling puzzle

Group 32

Alexia Dormann, Mariia Eremina and Valgerdur Jónsdóttir

Segmentation



Input image

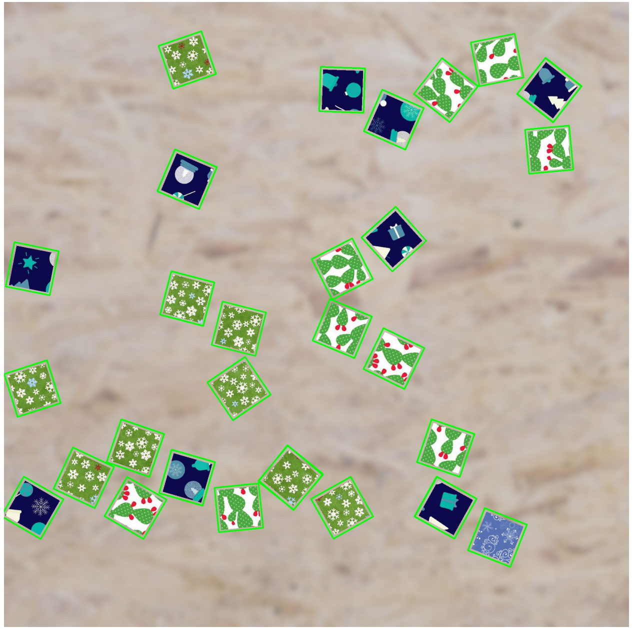


Image with segmentation lines

Segmentation

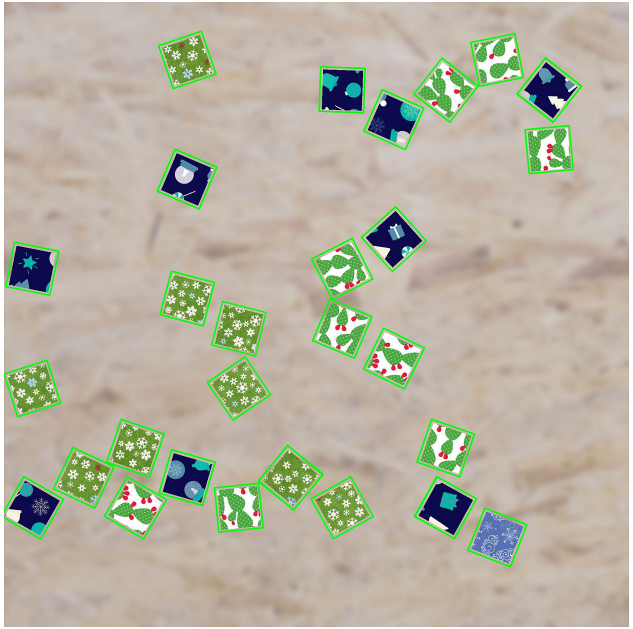
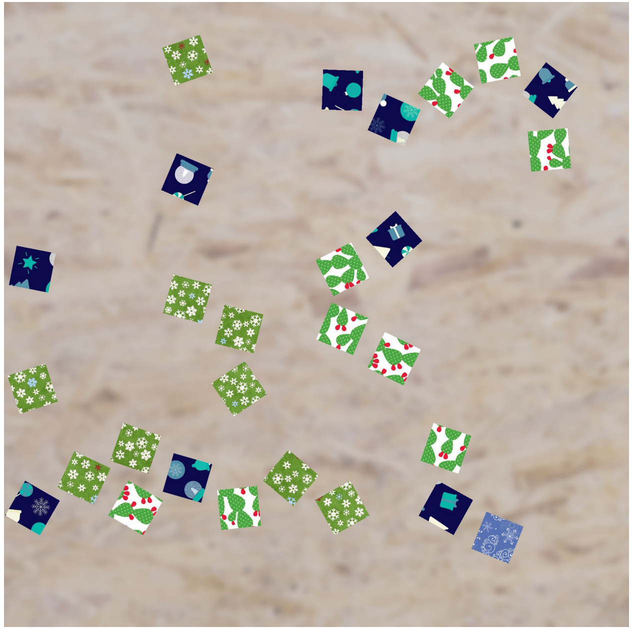


Image with segmentation lines

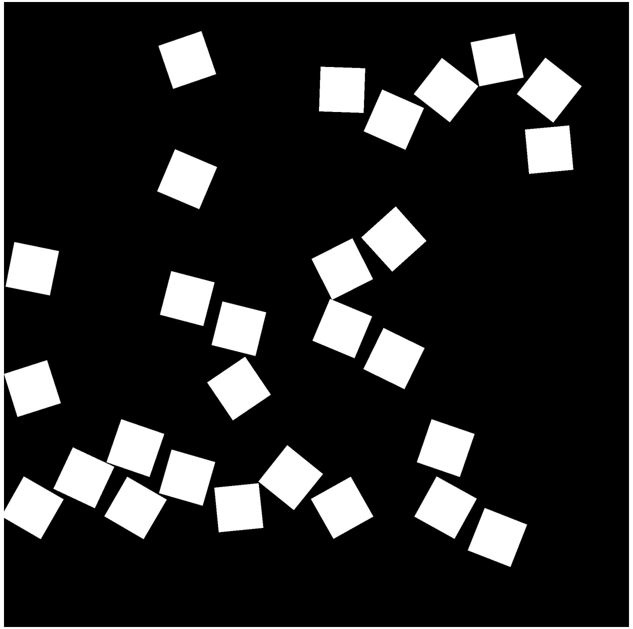
Segmentation algorithm:

- *Preprocessing:* median blur to remove small details on background and puzzle pieces
- *Edges detection* using canny edge detector
- *Dilate edges* to make them easier to detect
- *Fill in contours* to remove edges detected inside pieces
- *Find minimum area rectangles* fitting in contours
- *Define contours of pieces* as contour of the rectangle

Segmentation



Input image



Segmented image

Puzzle pieces extraction

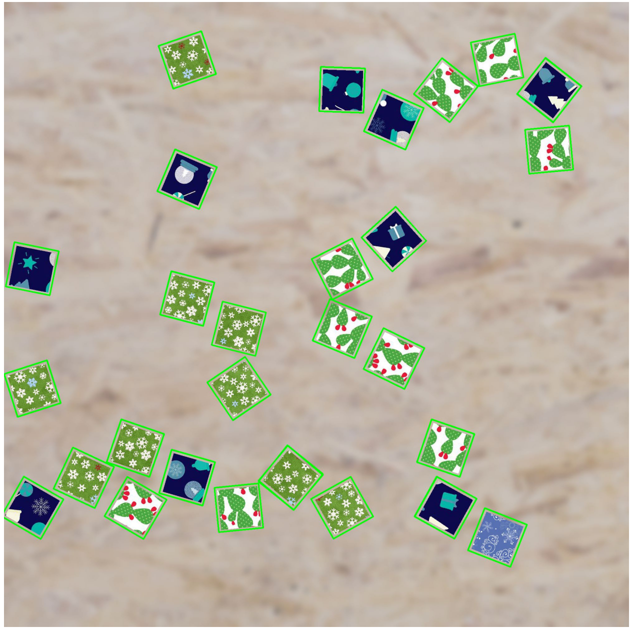
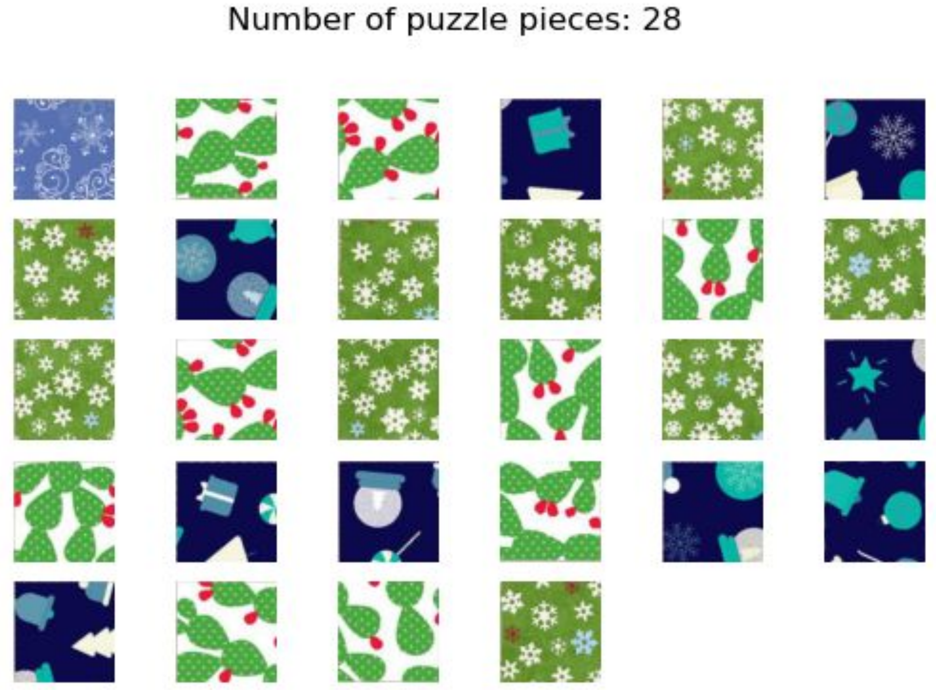


Image with segmentation lines

Extraction algorithm:

- *Fitting minimum rectangle area* in contours
- *Create blank image with only puzzle piece* for each contour
- *Rotate image* using center and angle of rectangle
- *Crop image to puzzle piece dimensions (128x128)*

Puzzle pieces extraction



Extracted puzzle pieces from the input image

Feature extraction

- **Color features**
  - Color histograms
  - Average and standard deviation of color values

Feature extraction

- **Color features**
  - Color histograms
  - Average and standard deviation of color values

- **Texture features**
  - Mean and standard deviation of filter response
  - Kurtosis of filter response
  - Power spectrum: Mean, max, standard deviation, etc.

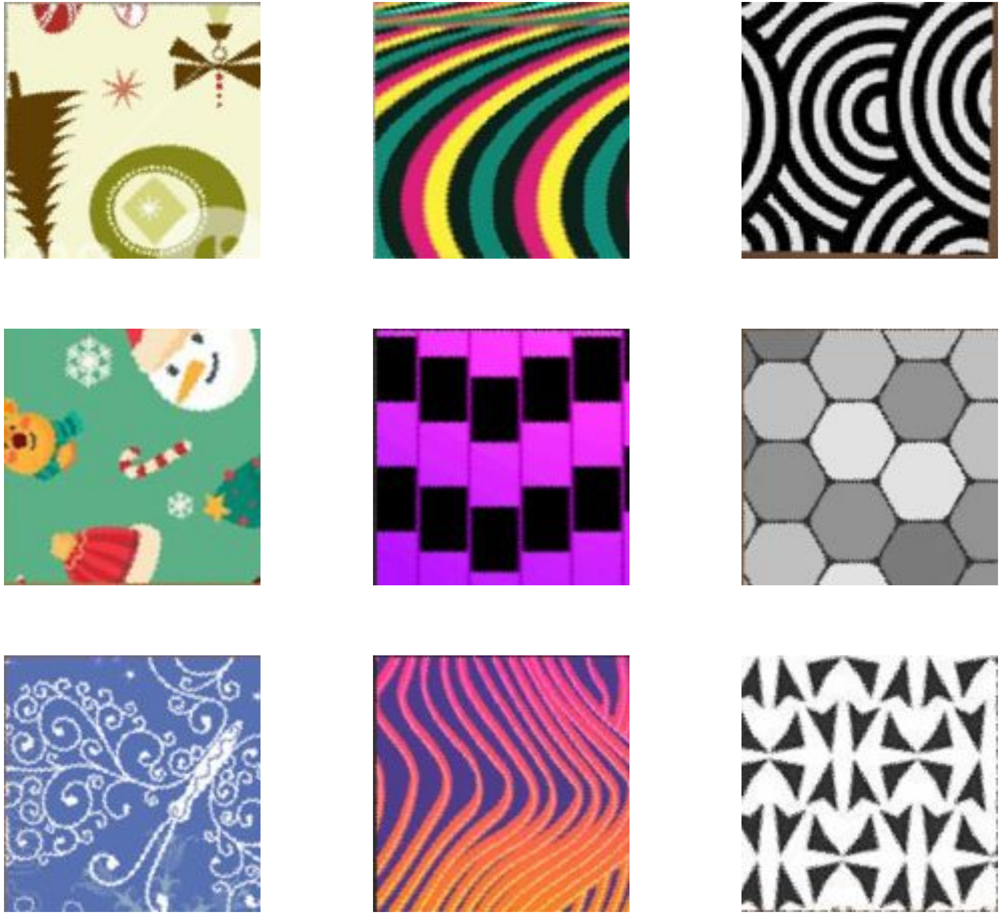
$$gb(x,y) = \exp\left(-\frac{1}{2}\left(\frac{x_\theta^2}{\sigma^2} + \frac{y_\theta^2}{(\Gamma\sigma)^2}\right)\right)\cos\left(\frac{2\pi}{\lambda}x_\theta + \psi\right)$$






Feature extraction

- **Color features**
  - Color histograms
  - Average and standard deviation of color values
- **Texture features**
  - Mean and standard deviation of filter response
  - Kurtosis of filter response
  - Power spectrum: Mean, max, standard deviation, etc.
- **Shape features**
  - Average circularity
  - Average area
  - Average perimeter



Feature selection using mutual information

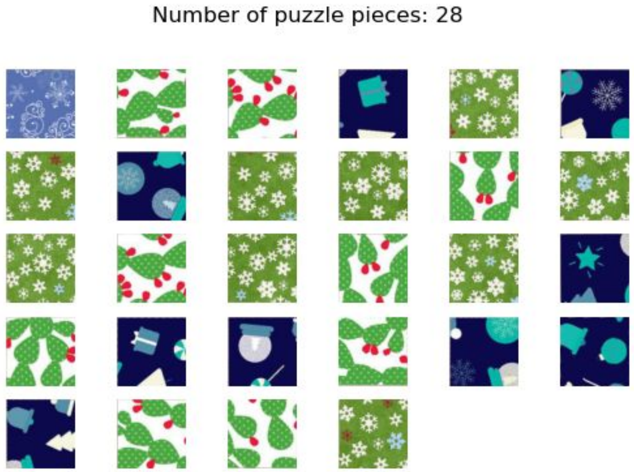
$$I(X;Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x,y) \log_2 \left(\frac{P(x,y)}{P(x)\,P(y)} \right)$$

MI tests features' ability to separate two classes.

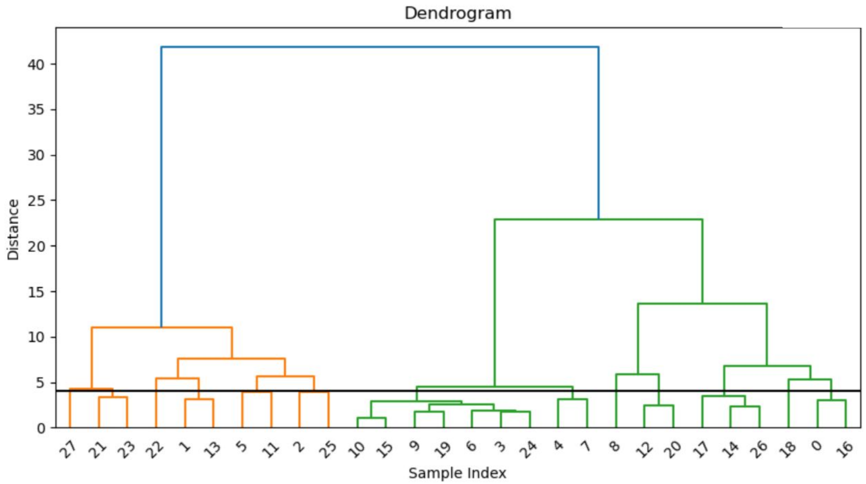
$$\underset{\text{\#pieces x 80}}{\begin{bmatrix} & & \\ & \cdots & \\ & & \end{bmatrix}} \longrightarrow \underset{\text{\#pieces x 45}}{\begin{bmatrix} & & \\ & \cdots & \\ & & \end{bmatrix}}$$

Clustering: Divisive "top-down" Hierarchical clustering

`scipy.cluster.hierarchy.dendrogram`

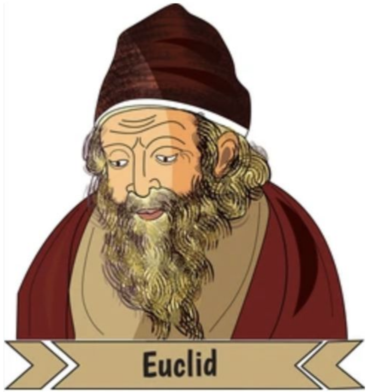


Extracted puzzle pieces from the input image



Dendrogram of the clustered pieces

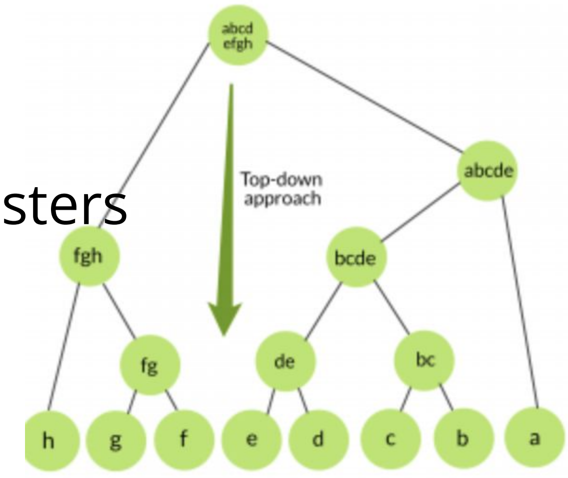
Hierarchical clustering



1. Assign all puzzle to a single class
2. Compute distance matrix across all pairs of data points
3. Split the cluster : linkage criterion

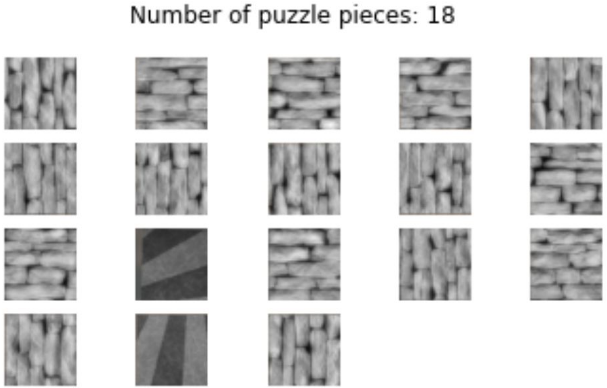
   Example : Single linkage $\boxed{\min_{a\in A,\, b\in B} d(a,b)}$

4. Update distance matrix to reflect new subclasters
5. Repeat and Update step 3 and 4

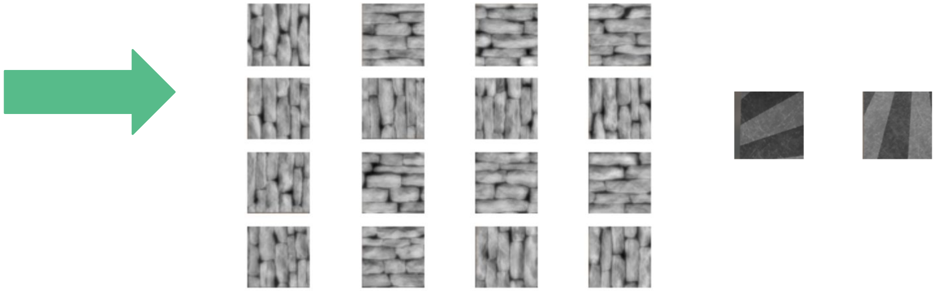


Hierarchical divisive
clustering

Clustering: Find outliers



Cluster of pieces with two outliers



Correctly clustered piece

Solving the puzzle

Shuffled puzzle

Solved puzzle

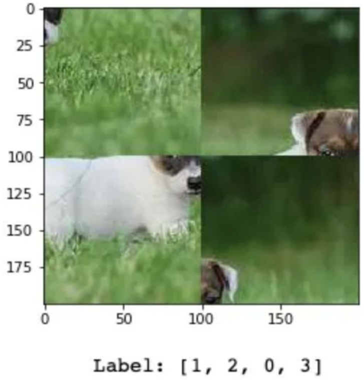
1. Solving JigSaw Puzzle Using Neural Nets base on permutation Invariance

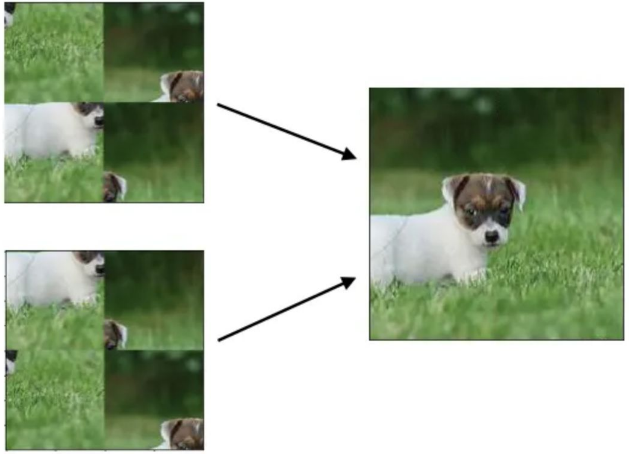
What is Permutation Invariance?

A function is a permutation invariant if its output does not change by changing the ordering of its input :

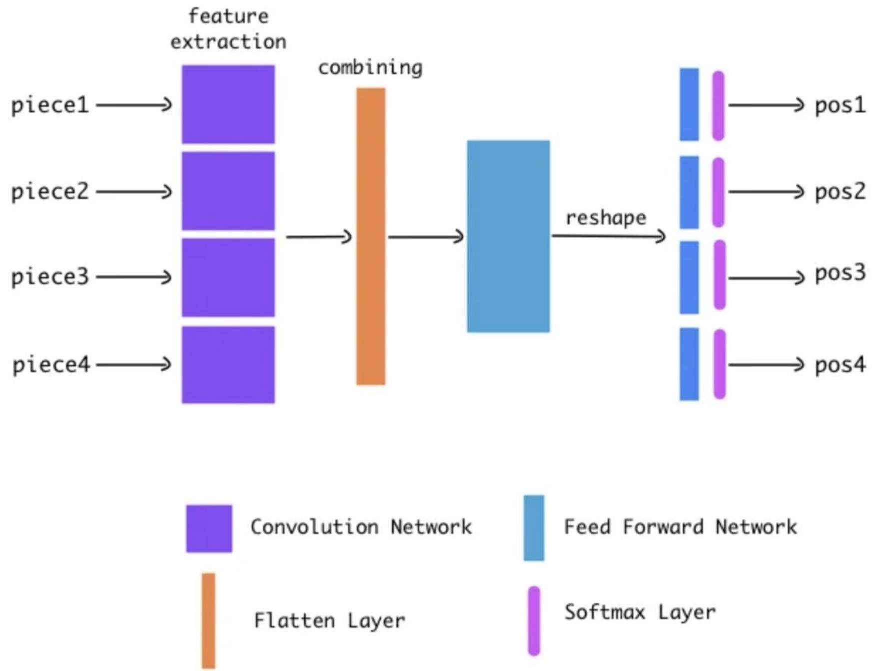
```
f(x,y,z) = xyz
```

```
2x2 puzzle = 4! = 24 combinations
3x3 puzzle = 9! = 362880 comb’ns
```





1. Neural Net Architecture




```
model = keras.models.Sequential()

model.add(td(ZeroPadding2D(2), input_shape

model.add(td(Conv2D(50, kernel_size=(5,5),
model.add(td(BatchNormalization()))
model.add(td(MaxPooling2D()))

model.add(td(Conv2D(100, kernel_size=(5,5)
model.add(td(BatchNormalization()))
model.add(td(Dropout(0.3)))

model.add(td(Conv2D(100, kernel_size=(3,3)
model.add(td(BatchNormalization()))
model.add(td(Dropout(0.3)))

model.add(td(Conv2D(200, kernel_size=(3,3)
model.add(td(BatchNormalization()))
model.add(td(Dropout(0.3)))

model.add(Flatten())  # combining all the

model.add(Dense(600, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(400, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.3))
model.add(Dense(16))
model.add(Reshape((4, 4)))          # reshap
model.add(Activation('softmax'))  # softma
```

2. Solving the puzzle using Graph-Based Puzzle Assembly Algorithm

Find neighborhood and rotation that
minimize difference between border pixels

Results

Train example # 2

Results

Train example # 5

Useful libraries

- **Segmentation:**
  - OpenCV (cv2)
  - NumPy
- **Feature extraction and selection:**
  - OpenCV (cv2)
  - Scikit-image and Scikit-learn
  - SciPy
  - Pandas
  - Matplotlib
  - NumPy
- **Clustering:**
  - SciPy
  - Scikit-image and Scikit-learn
  - NumPy