

Lecture 2: Image Classification

Assignment 1 Released

- <http://web.eecs.umich.edu/~justincj/teaching/eecs498/assignment1.html>
- Uses Python, PyTorch, and Google Colab
- Introduction to PyTorch Tensors
- K-Nearest Neighbor classification
- Due **Sunday September 15, 11:59pm EDT**

Assignment 1 Released

Q1: PyTorch 101 (50 points)

 Open in Colab

The notebook **pytorch101.ipynb** will walk you through the basics of working with tensors in PyTorch.

Q2: k-Nearest Neighbor classifier (50 points)

 Open in Colab

Click Here



The notebook **kNN.ipynb** will walk you through implementing a kNN classifier.

Google Colab: Cloud Computing in the Browser

The screenshot shows the Google Colab interface. At the top, there's a toolbar with various icons for file operations like 'New', 'Open', 'Save', and 'Run'. Below the toolbar is a navigation bar with 'kNN.ipynb' and other tabs like 'colab.research.google.com'. On the left, there's a sidebar with a 'File' menu and a 'Viewing' section. The main content area displays a Jupyter notebook titled 'EECS 498-007/598-005 Assignment 1-2: K-Nearest Neighbors (k-NN)'. The notebook contains text about implementing a K-Nearest Neighbors classifier on the CIFAR-10 dataset, followed by a code cell starting with 'Your Answer:'. To the right of the notebook, there's a list of bullet points under the heading 'Install starter code'.

EECS 498-007/598-005 Assignment 1-2: K-Nearest Neighbors (k-NN)

Before we start, please put your name and UMID in following format:
Firstname LASTNAME, #00000000 // e.g.) Justin JOHNSON, #12345678

Your Answer:
Hello WORLD, #XXXXXXX

K-Nearest Neighbors (k-NN)

In this notebook you will implement a K-Nearest Neighbors classifier on the [CIFAR-10 dataset](#). Recall that the K-Nearest Neighbor classifier does the following:

- During training, the classifier simply memorizes the training data
- During testing, test images are compared to each training image; the predicted label is the majority vote among the K nearest training examples.

After implementing the K-Nearest Neighbor classifier, you will use [cross-validation](#) to find the best value of K. The goals of this exercise are to go through a simple example of the data-driven image classification pipeline, and also to practice writing efficient, vectorized code in [PyTorch](#).

► **Install starter code**

We have implemented some utility functions for this exercise in the `cotools` package. Run this cell to download and install it.

```
[ ] !pip install git+https://github.com/deepvision-class/starter-code
```

Google Colab: Cloud Computing in the Browser

The screenshot shows the Google Colab interface. At the top, there's a toolbar with various icons for file operations like Open, Save, and Print. Below the toolbar is a navigation bar with back, forward, and search buttons, along with a link to the research.google.com drive.

The main area displays a notebook titled "KNN.ipynb". The code cell contains the following Python code:

```
[ ] !pip install git+https://github.com/deeplearning-class/starter-code
```

Below the code cell, a large callout box with a black arrow points from the text "Save a copy of the notebook to your Drive" to the "Save a copy in Drive..." button in the menu bar.

The menu bar includes options like File, Edit, View, Insert, Runtime, Tools, Help, and a "Locate in Drive" option. The "File" menu is currently open, showing options such as "Open notebook...", "Upload notebook...", "Rename...", "Move to trash", "Save a copy in Drive...", "Save a copy as a GitHub Gist...", "Save a copy in GitHub...", "Save and pin revision", "Revision history", "Download ipynb", "Download .py", "Update Drive preview", and "Print".

The right side of the screen shows a sidebar with sharing options, a viewing mode switch, and a list of recent notebooks including "New Python 3 notebook", "New Python 2 notebook", and "Assignment 1-2: K-Nearest Neighbors (k-NN)".

Justin Johnson

Lecture 2 - 5

September 9, 2019

Google Colab: Cloud Computing in the Browser

The screenshot shows the Google Colab interface. At the top, there's a toolbar with icons for file operations like Open, Save, and Print, along with a share button and a viewing mode switch. Below the toolbar is a menu bar with File, Edit, View, Insert, Runtime, Tools, and Help. The main area displays a notebook titled "kNN.ipynb". The code cell contains the following Python code:

```
[ ] !pip install git+https://github.com/deepvision-class/starter-code
```

Below the code cell, a note says: "We have implemented some utility functions for this exercise in the `utils` package. Run this cell to download and install it."

On the right side of the screen, there's a sidebar with options like "Download .ipynb", "Download .py", "Update Drive preview", and "Print". A large black arrow points from the text "when you are done" to the "Download .ipynb" button.

Justin Johnson

Lecture 2 - 6

September 9, 2019

Image Classification: A core computer vision task

Input: image



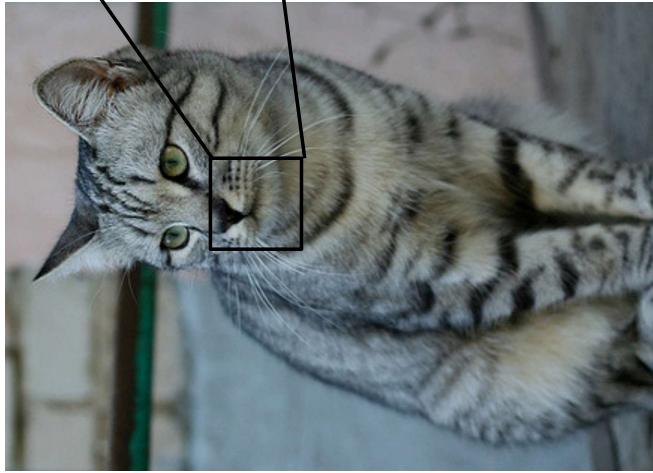
Output: Assign image to one
of a fixed set of categories

cat
bird
deer
dog
truck



This image by Nikita is
licensed under CC-BY 2.0

Problem: Semantic Gap



This image by Nikita is
licensed under [CC-BY 2.0](#)

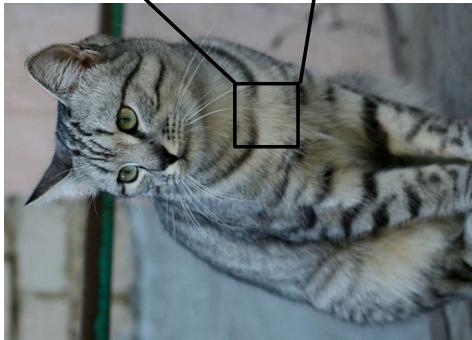
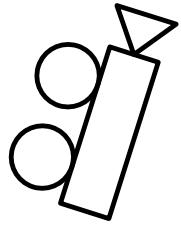
[1105 112 108 111 104 99 106 99 96 103 112 119 104 97 93 87]
[91 98 102 106 104 79 98 103 99 105 123 136 110 105 94 85]
[76 85 90 105 126 105 87 96 95 99 115 112 106 103 99 85]
[99 81 81 93 120 131 127 110 95 98 102 99 96 93 101 94]
[106 91 61 64 69 91 88 85 101 107 109 98 75 84 96 95]
[114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]
[133 137 147 103 65 81 80 65 52 54 74 84 102 93 85 82]
[128 137 144 149 109 95 96 70 62 65 63 63 60 73 96 101]
[125 133 148 137 119 121 117 94 65 79 88 65 54 64 72 98]
[127 125 131 147 133 127 126 131 111 96 89 75 61 64 72 84]
[115 114 109 123 150 148 131 118 113 109 100 92 74 65 72 78]
[89 93 90 97 108 147 131 118 113 114 113 109 106 95 77 80]
[63 77 86 81 77 79 102 123 117 115 117 125 130 115 87]
[62 65 82 89 78 71 80 101 124 126 119 101 107 114 131 119]
[63 65 75 88 89 71 62 81 120 138 135 105 81 98 110 118]
[87 65 71 87 106 95 69 45 76 130 126 107 92 94 105 112]
[118 97 82 6 117 123 116 16 41 51 95 93 89 95 102 107]
[164 146 112 80 82 120 124 104 76 48 45 66 88 101 102 109]
[157 170 157 120 93 86 114 132 112 97 69 55 70 82 99 94]
[130 128 134 161 139 100 109 118 121 134 114 87 65 53 69 86]
[128 112 96 117 158 144 120 115 104 107 102 93 87 81 72 79]
[123 107 96 86 83 112 152 149 122 109 104 75 80 107 112 99]
[122 121 102 80 82 86 94 117 148 153 102 58 78 92 107]
[122 164 148 103 71 56 78 83 93 103 119 139 102 61 69 841]

What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. $800 \times 600 \times 3$
(3 channels RGB)

Challenges: Viewpoint Variation



[1195 112 148 141 104 99 106 99 96 103 112 119 104 97 93 87]
[1196 113 149 142 105 99 107 98 96 104 113 120 105 98 94 88]
[1197 114 150 143 106 99 108 97 96 105 115 121 106 99 95 89]
[1198 115 151 144 107 99 109 98 95 106 116 122 107 99 96 90]
[1199 116 152 145 108 99 110 99 95 107 117 123 108 99 97 91]
[1200 117 153 146 109 99 111 100 96 108 118 124 109 99 98 92]
[1201 118 154 147 110 99 112 101 97 109 119 125 110 99 99 93]
[1202 119 155 148 111 100 113 102 98 110 120 126 111 100 100 94]
[1203 120 156 149 112 101 114 103 99 111 121 127 112 101 101 95]
[1204 121 157 150 113 102 115 104 100 112 122 128 113 102 102 96]
[1205 122 158 151 114 103 116 105 101 113 123 129 114 103 103 97]
[1206 123 159 152 115 104 117 106 102 114 124 130 115 104 104 98]
[1207 124 160 153 116 105 118 107 103 115 125 131 116 105 105 99]
[1208 125 161 154 117 106 119 108 104 116 126 132 117 106 106 100]
[1209 126 162 155 118 107 120 109 105 117 127 133 118 107 107 101]
[1210 127 163 156 119 108 121 110 106 118 128 134 119 108 108 102]
[1211 128 164 157 120 109 122 111 107 119 129 135 120 109 109 103]
[1212 129 165 158 121 110 123 112 108 120 130 136 121 110 110 104]
[1213 130 166 159 122 111 124 113 109 121 131 137 122 111 111 105]
[1214 131 167 160 123 112 125 114 110 122 132 138 123 112 112 106]
[1215 132 168 161 124 113 126 115 111 123 133 139 124 113 113 107]
[1216 133 169 162 125 114 127 116 112 124 134 140 125 114 114 108]
[1217 134 170 163 126 115 128 117 113 125 135 141 126 115 115 109]
[1218 135 171 164 127 116 129 118 114 126 136 142 127 116 116 110]
[1219 136 172 165 128 117 130 119 115 127 137 143 128 117 117 111]
[1220 137 173 166 129 118 131 120 116 128 138 144 129 118 118 112]
[1221 138 174 167 130 119 132 121 117 129 139 145 130 119 119 113]
[1222 139 175 168 131 120 133 122 118 130 140 146 131 120 120 114]
[1223 140 176 169 132 121 134 123 119 131 141 147 132 121 121 115]
[1224 141 177 170 133 122 135 124 120 132 142 148 133 122 122 116]
[1225 142 178 171 134 123 136 125 121 133 143 149 134 123 123 117]
[1226 143 179 172 135 124 137 126 122 134 144 150 135 124 124 118]
[1227 144 180 173 136 125 138 127 123 135 145 151 136 125 125 119]
[1228 145 181 174 137 126 139 128 124 136 146 152 137 126 126 120]
[1229 146 182 175 138 127 140 129 125 137 147 153 138 127 127 119]
[1230 147 183 176 139 128 141 130 126 138 148 154 139 128 128 121]
[1231 148 184 177 140 129 142 131 127 139 149 155 140 129 129 122]
[1232 149 185 178 141 130 143 132 128 140 150 156 141 130 130 123]
[1233 150 186 179 142 131 144 133 129 141 151 157 142 131 131 124]
[1234 151 187 180 143 132 145 134 130 142 152 158 143 132 132 125]
[1235 152 188 181 144 133 146 135 131 143 153 159 144 133 133 126]
[1236 153 189 182 145 134 147 136 132 144 154 160 145 134 134 127]
[1237 154 190 183 146 135 148 137 133 145 155 161 146 135 135 128]
[1238 155 191 184 147 136 149 138 134 146 156 162 147 136 136 129]
[1239 156 192 185 148 137 150 139 135 147 157 163 148 137 137 130]
[1240 157 193 186 149 138 151 140 136 148 158 164 149 138 138 131]
[1241 158 194 187 150 139 152 141 137 149 159 165 150 140 139 132]
[1242 159 195 188 151 140 153 142 138 150 160 166 151 141 140 133]
[1243 160 196 189 152 141 154 143 139 151 161 167 152 142 141 134]
[1244 161 197 190 153 142 155 144 140 152 162 168 153 143 142 135]
[1245 162 198 191 154 143 156 145 141 153 163 169 154 144 143 136]
[1246 163 199 192 155 144 157 146 142 154 164 170 155 145 144 137]
[1247 164 200 193 156 145 158 147 143 155 165 171 156 146 145 138]
[1248 165 201 194 157 146 159 148 144 156 166 172 157 147 146 139]
[1249 166 202 195 158 147 160 149 145 157 167 173 158 148 147 140]
[1250 167 203 196 159 148 161 150 146 158 168 174 159 149 148 141]
[1251 168 204 197 160 149 162 151 147 159 169 175 160 150 149 142]
[1252 169 205 198 161 150 163 152 148 160 170 176 161 151 150 143]
[1253 170 206 199 162 151 164 153 149 161 171 177 162 152 151 144]
[1254 171 207 200 163 152 165 154 150 162 172 178 163 153 152 145]
[1255 172 208 201 164 153 166 155 151 163 173 179 164 154 153 146]
[1256 173 209 202 165 154 167 156 152 164 174 180 165 155 154 147]
[1257 174 210 203 166 155 168 157 153 165 175 181 166 156 155 148]
[1258 175 211 204 167 156 169 158 154 166 176 182 167 157 156 149]
[1259 176 212 205 168 157 170 159 155 167 177 183 168 158 157 150]
[1260 177 213 206 169 158 171 160 156 168 178 184 169 159 158 151]
[1261 178 214 207 170 159 172 161 157 169 179 185 170 160 159 152]
[1262 179 215 208 171 160 173 162 158 170 180 186 171 161 160 153]
[1263 180 216 209 172 161 174 163 159 171 181 187 172 162 161 154]
[1264 181 217 210 173 162 175 164 160 172 182 188 173 163 162 155]
[1265 182 218 211 174 163 176 165 161 173 183 189 174 164 163 156]
[1266 183 219 212 175 164 177 166 162 174 184 190 175 165 164 157]
[1267 184 220 213 176 165 178 167 163 175 185 191 176 166 165 158]
[1268 185 221 214 177 166 179 168 164 176 186 192 177 167 166 159]
[1269 186 222 215 178 167 180 169 165 177 187 193 178 168 167 160]
[1270 187 223 216 179 168 181 170 166 178 188 194 179 169 168 161]
[1271 188 224 217 180 169 182 171 167 179 189 195 180 170 169 162]
[1272 189 225 218 181 170 183 172 168 180 190 196 181 171 170 163]
[1273 190 226 219 182 171 184 173 169 181 191 197 182 172 171 164]
[1274 191 227 220 183 172 185 174 170 182 192 198 183 173 172 165]
[1275 192 228 221 184 173 186 175 171 183 193 200 184 174 173 166]
[1276 193 229 222 185 174 187 176 172 184 194 201 185 175 174 167]
[1277 194 230 223 186 175 188 177 173 185 195 202 186 176 175 168]
[1278 195 231 224 187 176 189 178 174 186 196 203 187 177 176 169]
[1279 196 232 225 188 177 190 179 175 187 197 204 188 178 177 170]
[1280 197 233 226 189 178 191 180 176 188 198 205 189 179 178 171]
[1281 198 234 227 190 179 192 181 177 189 199 206 190 180 179 172]
[1282 199 235 228 191 180 193 182 178 190 200 207 191 181 180 173]
[1283 200 236 229 192 181 194 183 179 191 201 208 192 182 181 174]
[1284 201 237 230 193 182 195 184 180 192 202 209 193 183 182 175]
[1285 202 238 231 194 183 196 185 181 193 203 210 194 184 183 176]
[1286 203 239 232 195 184 197 186 182 194 204 211 195 185 184 177]
[1287 204 240 233 196 185 198 187 183 195 205 212 196 186 185 178]
[1288 205 241 234 197 186 199 188 184 196 206 213 197 187 186 179]
[1289 206 242 235 198 187 200 189 185 197 207 214 198 188 187 180]
[1290 207 243 236 199 188 201 190 186 198 208 215 199 189 188 181]
[1291 208 244 237 200 189 202 191 187 199 209 216 200 190 189 182]
[1292 209 245 238 201 190 203 192 188 200 210 217 201 191 190 183]
[1293 210 246 239 202 191 204 193 189 201 211 218 202 192 191 184]
[1294 211 247 240 203 192 205 194 190 202 212 219 203 193 192 185]
[1295 212 248 241 204 193 206 195 191 203 213 220 204 194 193 186]
[1296 213 249 242 205 194 207 196 192 204 214 221 205 195 194 187]
[1297 214 250 243 206 195 208 197 193 205 215 222 206 196 195 188]
[1298 215 251 244 207 196 209 198 194 206 216 223 207 197 196 190]
[1299 216 252 245 208 197 210 199 195 207 217 224 208 198 197 191]
[1300 217 253 246 209 198 211 200 196 208 218 225 209 199 198 192]
[1301 218 254 247 210 199 212 201 197 209 219 226 210 200 199 193]
[1302 219 255 248 211 200 213 202 198 210 220 227 211 201 200 194]
[1303 220 256 249 212 201 214 203 199 211 221 228 212 202 201 195]
[1304 221 257 250 213 202 215 204 200 212 222 229 213 203 202 196]
[1305 222 258 251 214 203 216 205 201 213 223 230 214 204 203 197]
[1306 223 259 252 215 204 217 206 202 214 224 231 215 205 204 198]
[1307 224 260 253 216 205 218 207 203 215 225 232 216 206 205 201]
[1308 225 261 254 217 206 219 208 204 216 226 233 217 207 206 202]
[1309 226 262 255 218 207 220 209 205 217 227 234 218 208 207 203]
[1310 227 263 256 219 208 221 210 206 218 228 235 219 209 208 204]
[1311 228 264 257 220 209 222 211 207 219 229 236 220 210 209 205]
[1312 229 265 258 221 210 223 212 208 220 230 237 221 211 210 206]
[1313 230 266 259 222 211 224 213 209 221 231 238 222 212 211 207]
[1314 231 267 260 223 212 225 214 210 222 232 239 223 213 212 208]
[1315 232 268 261 224 213 226 215 211 223 233 240 224 214 213 209]
[1316 233 269 262 225 214 227 216 212 224 234 241 225 215 214 210]
[1317 234 270 263 226 215 228 217 213 225 235 242 226 216 215 209]
[1318 235 271 264 227 216 229 218 214 226 236 243 227 217 216 211]
[1319 236 272 265 228 217 230 219 215 227 237 244 228 218 217 209]
[1320 237 273 266 229 218 231 220 216 228 238 245 229 219 218 212]
[1321 238 274 267 230 219 232 221 217 229 239 246 230 220 219 213]
[1322 239 275 268 231 220 233 222 218 230 240 247 231 221 220 216]
[1323 240 276 269 232 221 234 223 219 231 241 248 232 222 221 217]
[1324 241 277 270 233 222 235 224 220 232 242 249 233 223 222 218]
[1325 242 278 271 234 223 236 225 221 233 243 250 234 224 223 221]
[1326 243 279 272 235 224 237 226 222 234 244 251 235 225 224 222]
[1327 244 280 273 236 225 238 227 223 235 245 252 236 226 225 223]
[1328 245 281 274 237 226 239 228 224 236 246 253 237 227 226 224]
[1329 246 282 275 238 227 240 229 225 237 247 254 238 228 227 225]
[1330 247 283 276 239 228 241 230 226 238 248 255 239 229 228 226]
[1331 248 284 277 240 229 242 231 227 239 249 256 240 230 229 227]
[1332 249 285 278 241 230 243 232 228 240 250 257 241 231 230 228]
[1333 250 286 279 242 231 244 233 229 241 251 258 242 232 231 229]
[1334 251 287 280 243 232 245 234 230 242 252 259 243 233 232 230]
[1335 252 288 281 244 233 246 235 231 243 253 260 244 234 233 231]
[1336 253 289 282 245 234 247 236 232 244 254 261 245 235 234 232]
[1337 254 290 283 246 235 248 237 233 245 255 262 246 236 235 233]
[1338 255 291 284 247 236 249 238 234 246 256 263 247 237 236 234]
[1339 256 292 285 248 237 250 239 235 247 257 264 248 238 237 235]
[1340 257 293 286 249 238 251 240 236 248 258 265 249 239 238 236]
[1341 258 294 287 250 239 252 241 237 249 259 266 250 240 239 237]
[1342 259 295 288 251 240 253 242 238 250 260 267 251 241 240 238]
[1343 260 296 289 252 241 254 243 239 251 261 268 252 242 241 239]
[1344 261 297 290 253 242 255 244 240 252 262 269 253 243 242 240]
[1345 262 298 291 254 243 256 245 241 253 2

Challenges: Intraclass Variation



This image is [CC0 1.0](#) public domain

Challenges: Fine-Grained Categories

Maine Coon



This image is free for use under the Pixabay License

Ragdoll



This image is CC0 public domain

American Shorthair



This image is CC0 public domain

Challenges: Background Clutter



This image is [CC0 1.0](#) public domain



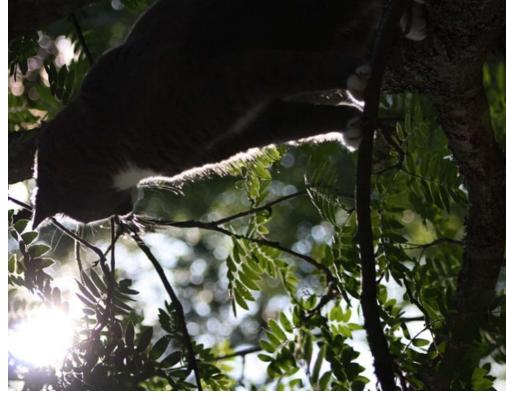
This image is [CC0 1.0](#) public domain

Justin Johnson

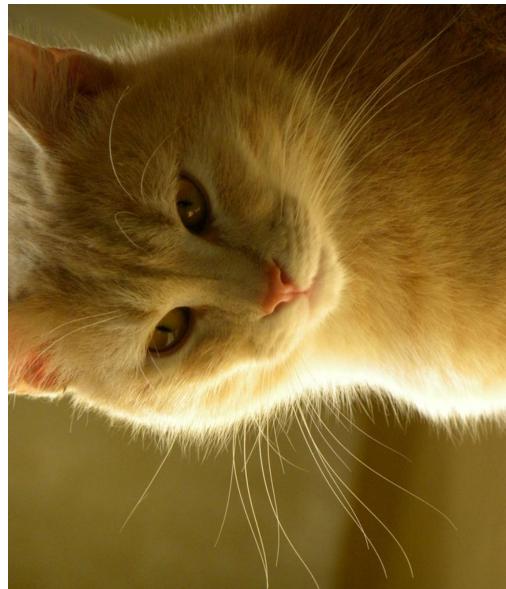
Lecture 2 - 12

September 9, 2019

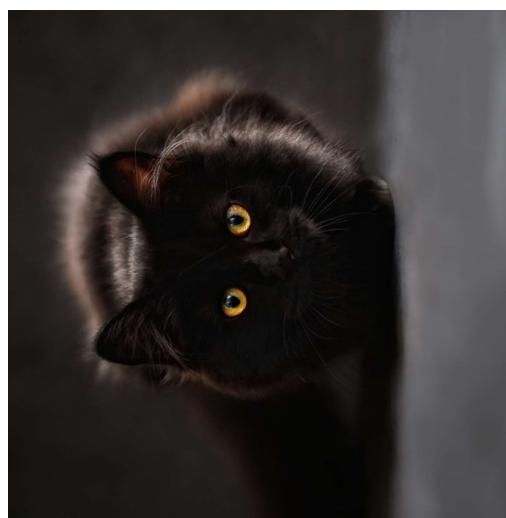
Challenges: Illumination Changes



[This image](#) is CC0 1.0 public domain



[This image](#) is CC0 1.0 public domain



[This image](#) is CC0 1.0 public domain



[This image](#) is CC0 1.0 public domain

Justin Johnson

Lecture 2 - 13

September 9, 2019

Challenges: Deformation



This image by Umberto Salvagnin is licensed under CC-BY 2.0



This image by Umberto Salvagnin is licensed under CC-BY 2.0



This image by sare bear is licensed under CC-BY 2.0



This image by Tom Thai is licensed under CC-BY 2.0

Challenges: Occlusion



This image by [jonsson](#) is licensed under [CC-BY 2.0](#)



This image is [CC0 1.0](#) public domain



This image is [CC0 1.0](#) public domain

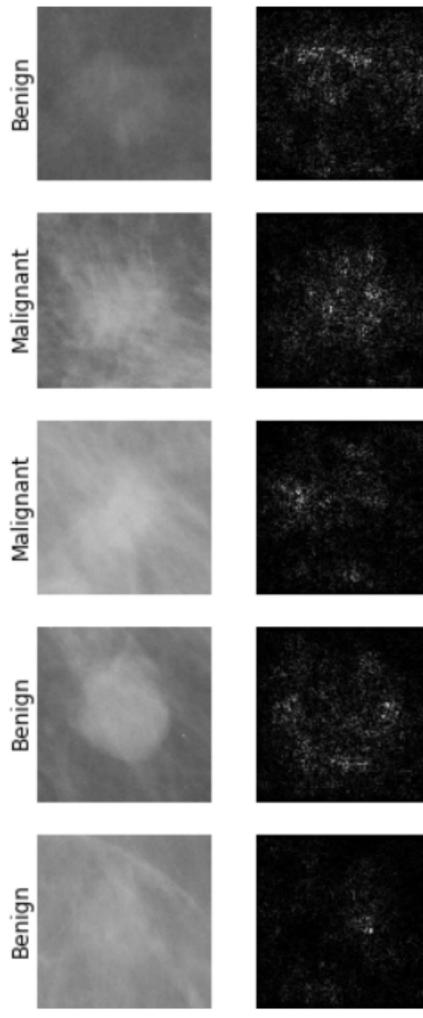
Justin Johnson

Lecture 2 - 15

September 9, 2019

Image Classification: Very Useful!

Medical Imaging



Levy et al, 2016

Figure reproduced with permission



Dieleman et al, 2014



From left to right: public domain by NASA, usage permitted by
ESA/Hubble, public domain by NASA, and public domain.

Kaggle Challenge

This image by Christin Khan is in the public domain and
originally came from the U.S. NOAA.

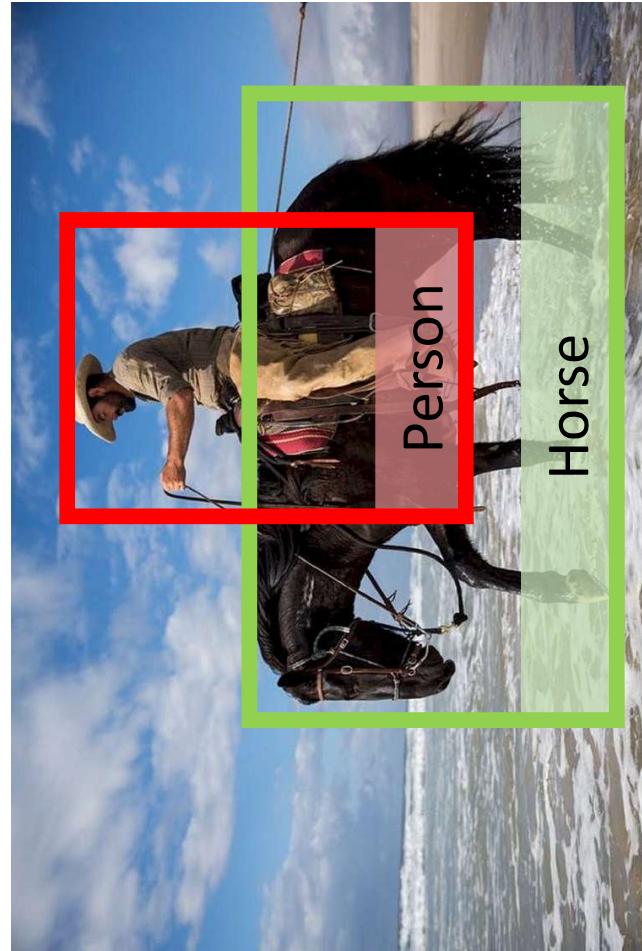
Justin Johnson

Lecture 2 - 16

September 9, 2019

Image Classification: Building Block for other tasks!

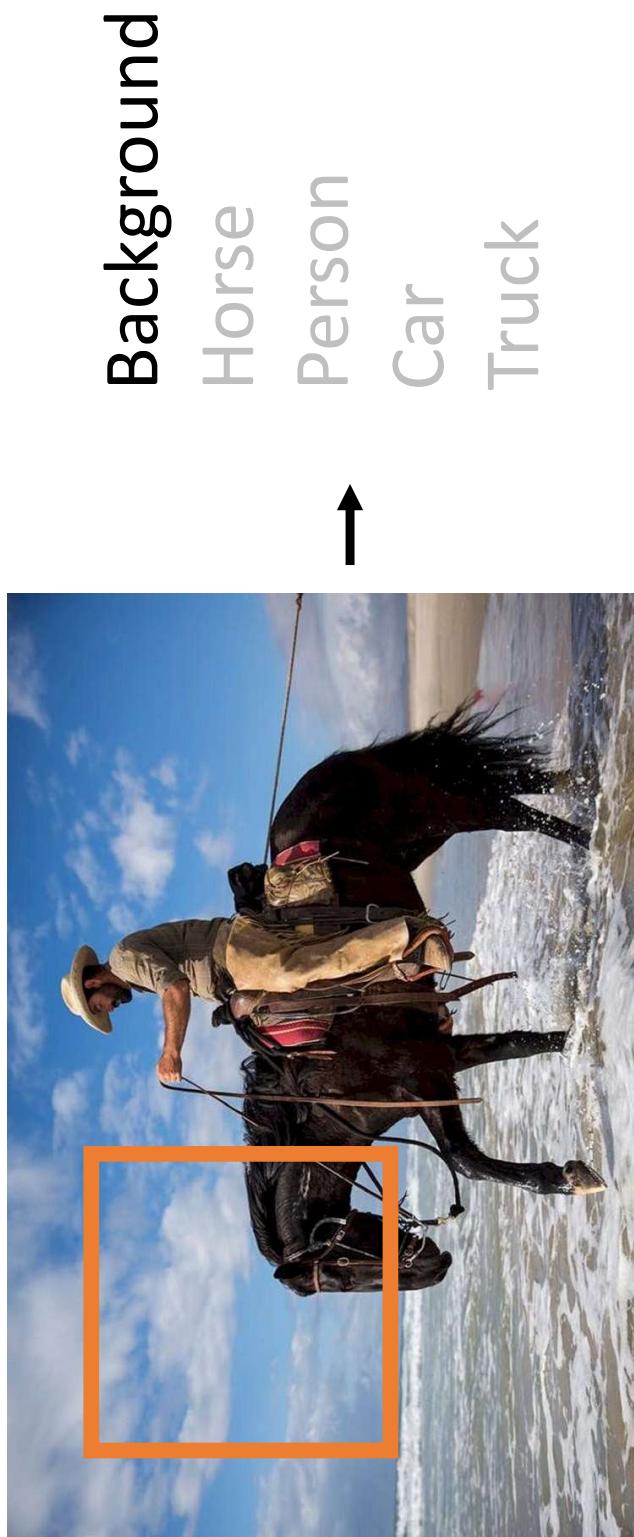
Example: Object Detection



This image is free to use under the Pexels license

Image Classification: Building Block for other tasks!

Example: Object Detection



This image is free to use under the Pexels license

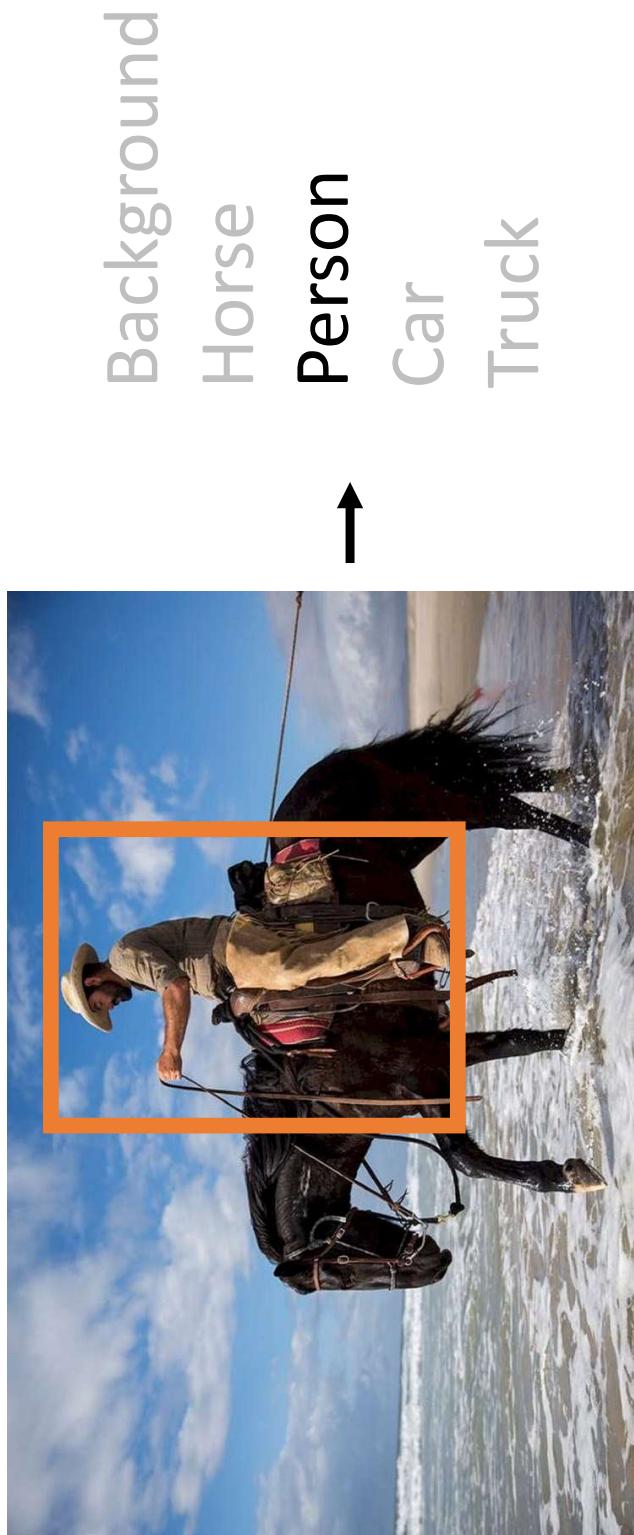
Justin Johnson

Lecture 2 - 18

September 9, 2019

Image Classification: Building Block for other tasks!

Example: Object Detection



This image is free to use under the Pexels license

Image Classification: Building Block for other tasks!

Example: Image Captioning



riding
cat
horse
→ man
when
Man
...
<STOP>

What word
to say next?

Caption:

This image is free to use under the Pexels license

Image Classification: Building Block for other tasks!

Example: Image Captioning



riding What word
cat to say next?

horse →
man when
...
<STOP>

This image is free to use under the Pexels license

Image Classification: Building Block for other tasks!

Example: Image Captioning



riding
cat
horse
man
when
...

What word
to say next?

Caption:
Man riding horse
<STOP>

A sequence of words and punctuation marks, with the word "horse" in bold. An arrow points from the word "horse" towards the image. Above the sequence, the question "What word to say next?" is asked. To the right, a caption is provided: "Caption: Man riding horse ... <STOP>".

This image is free to use under the Pexels license

Image Classification: Building Block for other tasks!

Example: Image Captioning



riding
cat
horse
man
when
...

What word
to say next?

Caption:
Man riding horse
<STOP>

A diagram illustrating the process of generating an image caption. On the left, a sequence of words is shown: "riding", "cat", "horse", "man", "when", followed by three dots. An arrow points from this sequence to the right. Above the sequence, the text "What word to say next?" is displayed. To the right of the arrow, the word "Caption:" is followed by the generated caption "Man riding horse". At the far right, the text "<STOP>" is enclosed in angle brackets.

This image is free to use under the Pexels license

Image Classification: Building Block for other tasks!

Example: Playing Go



Where to
play next?

(1, 1)
(1, 2)

...
→ (1, 19)

...
(19, 19)

This image is CC0 public domain

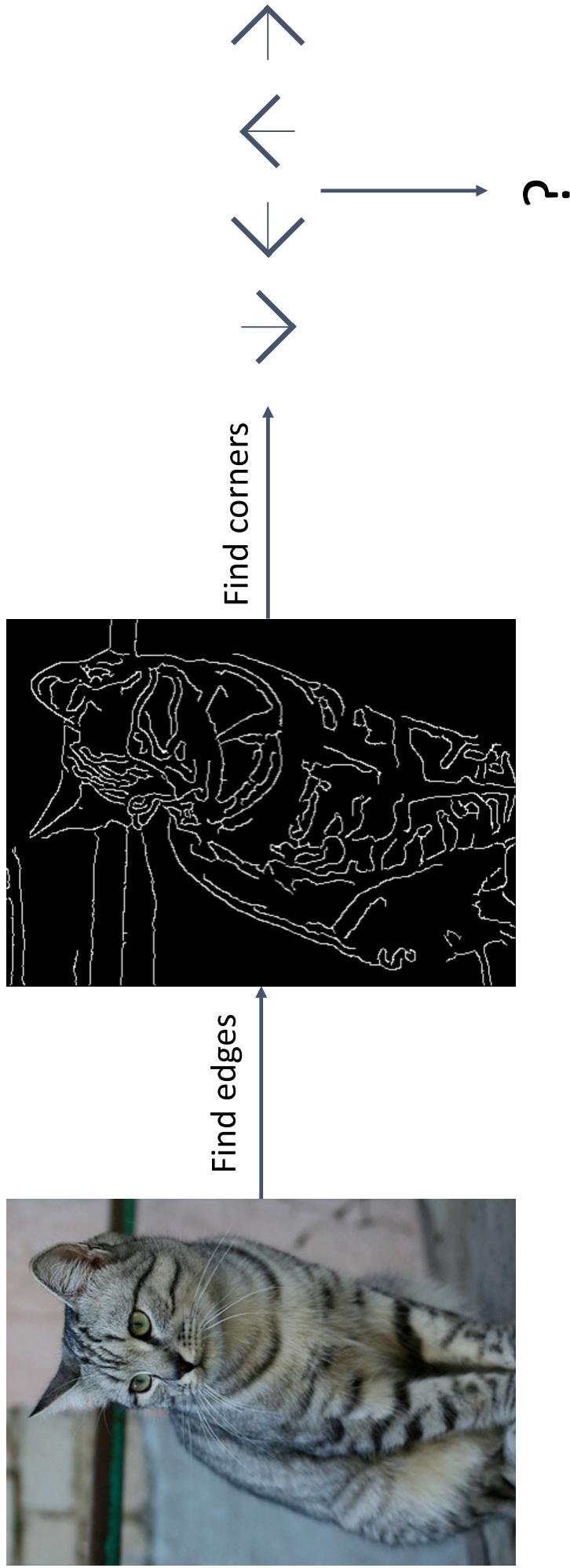
An Image Classifier

```
def classify_image(image):  
    # Some magic here?  
    return class_label
```

Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm
for recognizing a cat, or other classes.

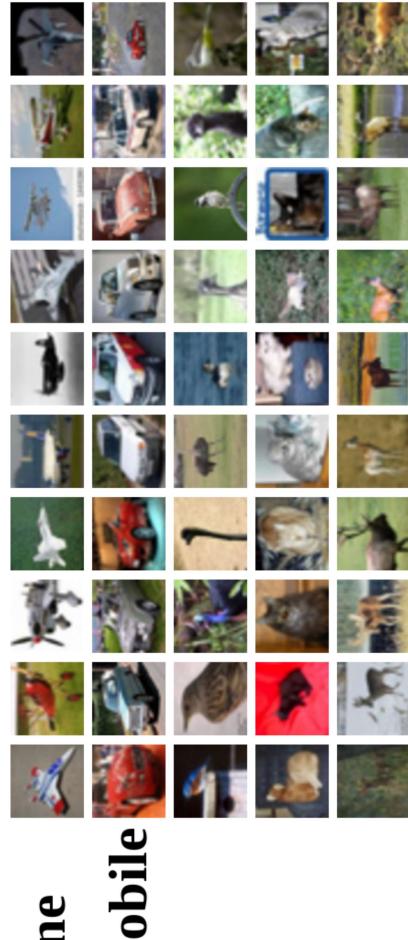
You could try ...



Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

Example training set



```
def train(images, labels):  
    # Machine learning!  
    return model  
  
def predict(model, test_images):  
    # Use model to predict labels  
    return test_labels
```

Image Classification Datasets: MNIST

3	3	5	7	5	0	4	8	6	1	7
9	6	4	6	3	0	1	3	3		
1	1	5	3	3	5	7	9	0	8	3
4	2	3	9	7	0	1	9	1	8	6
8	6	9	3	6	7	2	1	2	3	7
7	3	3	3	3	4	2	1	5	5	4
0	7	3	6	9	7	0	2	1	7	4
4	0	8	4	1	9	7	7	1	3	9
1	4	2	6	1	1	8	8	5	3	3
0	1	0	4	7	7	1	8	1	5	2
6	6	5	5	5	5	9	3	6	2	0
0	6	6	5	2	4	7	4	0	2	4
6	6	6	6	6	6	6	6	6	6	0

10 classes: Digits 0 to 9

28x28 grayscale images

50k training images

10k test images

Image Classification Datasets: MNIST

3	3	5	7	5	0	4	8	6	1	7
9	6	4	9	6	3	0	1	3	3	3
1	1	1	5	3	3	5	7	9	0	8
4	8	3	9	9	7	0	1	9	1	8
2	6	9	3	6	7	9	2	1	2	3
7	3	6	9	4	7	2	1	5	5	4
1	7	3	3	3	3	2	1	8	0	2
0.	8	4	1	9	7	7	7	8	1	7
9	5	6	2	8	5	3	2	1	5	3
0	2	4	1	6	9	0	0	8	0	9

10 classes: Digits 0 to 9
28x28 grayscale images
50k training images
10k test images

“Drosophila of computer vision”

Results from MNIST often do not hold on more complex datasets!

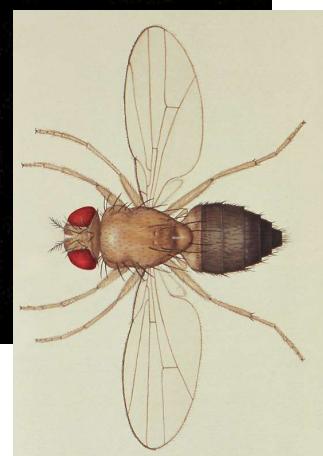
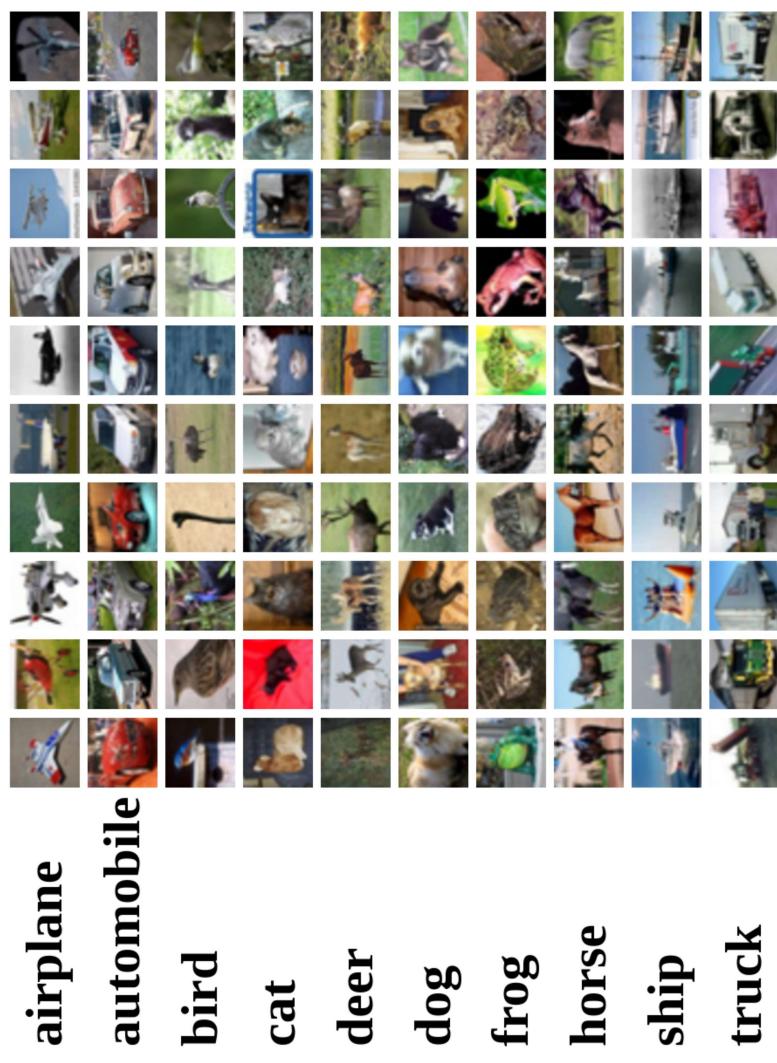


Image Classification Datasets: CIFAR10

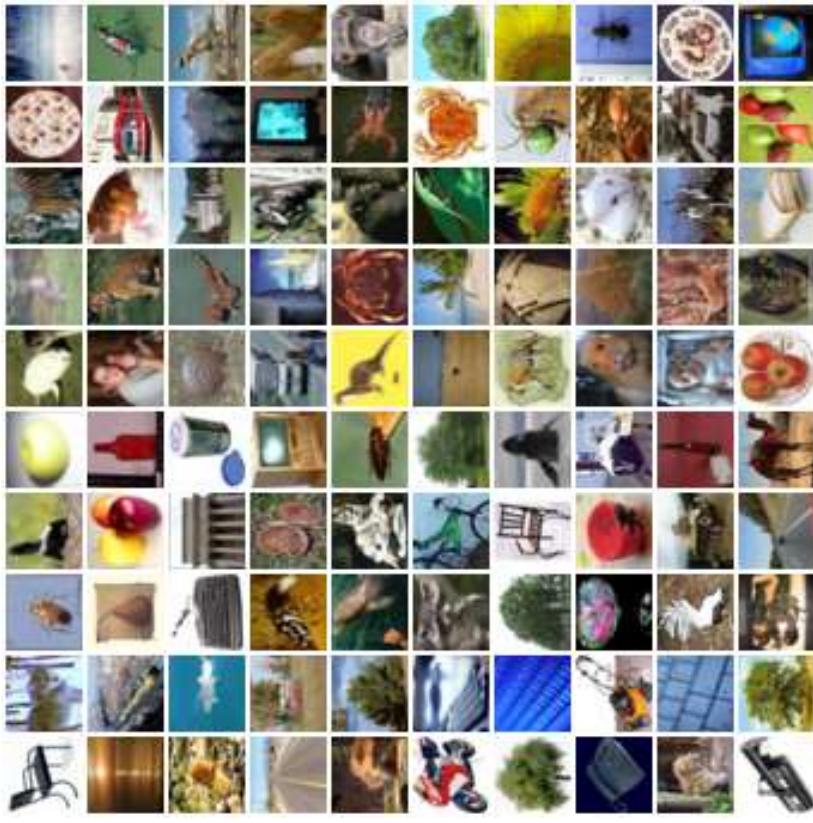


10 classes
50k training images (5k per class)
10k testing images (1k per class)
32x32 RGB images

We will use this dataset for
homework assignments

Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Technical Report, 2009.

Image Classification Datasets: CIFAR100



100 classes

50k training images (500 per class)

10k testing images (100 per class)

32x32 RGB images

20 superclasses with 5 classes each:

Aquatic mammals: beaver, dolphin,

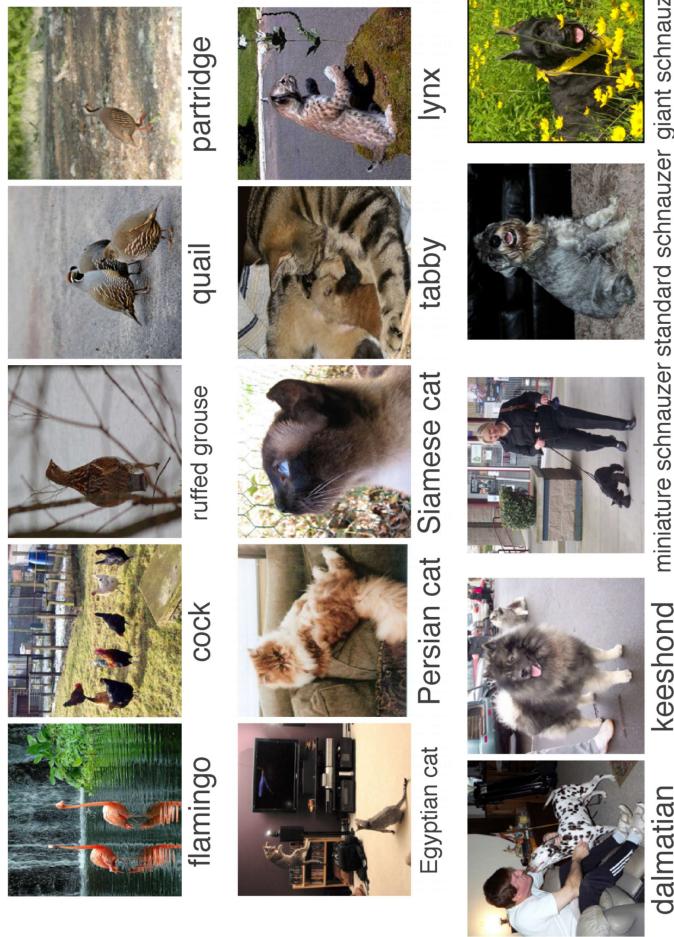
otter, seal, whale

Trees: Maple, oak, palm, pine, willow

Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Technical Report, 2009.

Image Classification Datasets: ImageNet

1000 classes



~1.3M training images (~1.3K per class)
50K validation images (50 per class)
100K test images (100 per class)

Performance metric: **Top 5 accuracy**

Algorithm predicts 5 labels for each image; one of them needs to be right

Deng et al, "ImageNet: A Large-Scale Hierarchical Image Database", CVPR 2009
Russakovsky et al, "ImageNet Large Scale Visual Recognition Challenge", IJCV 2015

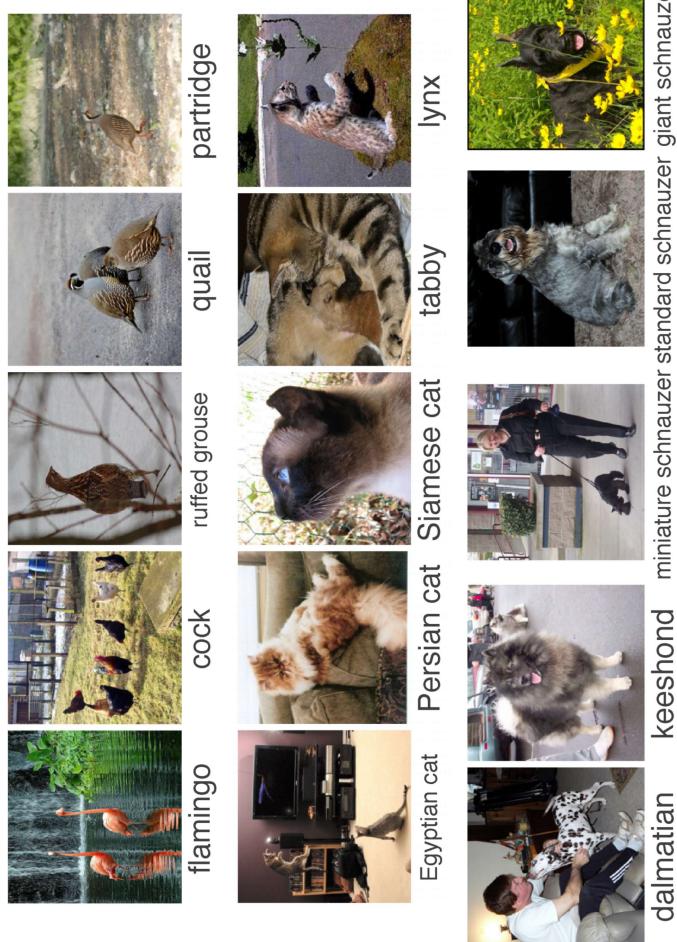
Justin Johnson

Lecture 2 - 32

September 9, 2019

Image Classification Datasets: ImageNet

1000 classes



~1.3M training images (~1.3K per class)
50K validation images (50 per class)
100K test images (100 per class)
test labels are secret!

Images have variable size, but often
resized to **256x256** for training

There is also a 22k category version of
ImageNet, but less commonly used

Deng et al, "ImageNet: A Large-Scale Hierarchical Image Database", CVPR 2009
Russakovsky et al, "ImageNet Large Scale Visual Recognition Challenge", IJCV 2015

Justin Johnson

Lecture 2 - 33

September 9, 2019

Image Classification Datasets: MIT Places



365 classes of different scene types

~8M training images
18.25K val images (50 per class)
328.5K test images (900 per class)

Images have variable size, often
resize to **256x256** for training

Zhou et al., "Places: A 10 million Image Database for Scene Recognition", TPAMI 2017

Justin Johnson

Lecture 2 - 34

September 9, 2019

Classification Datasets: Number of Training Pixels

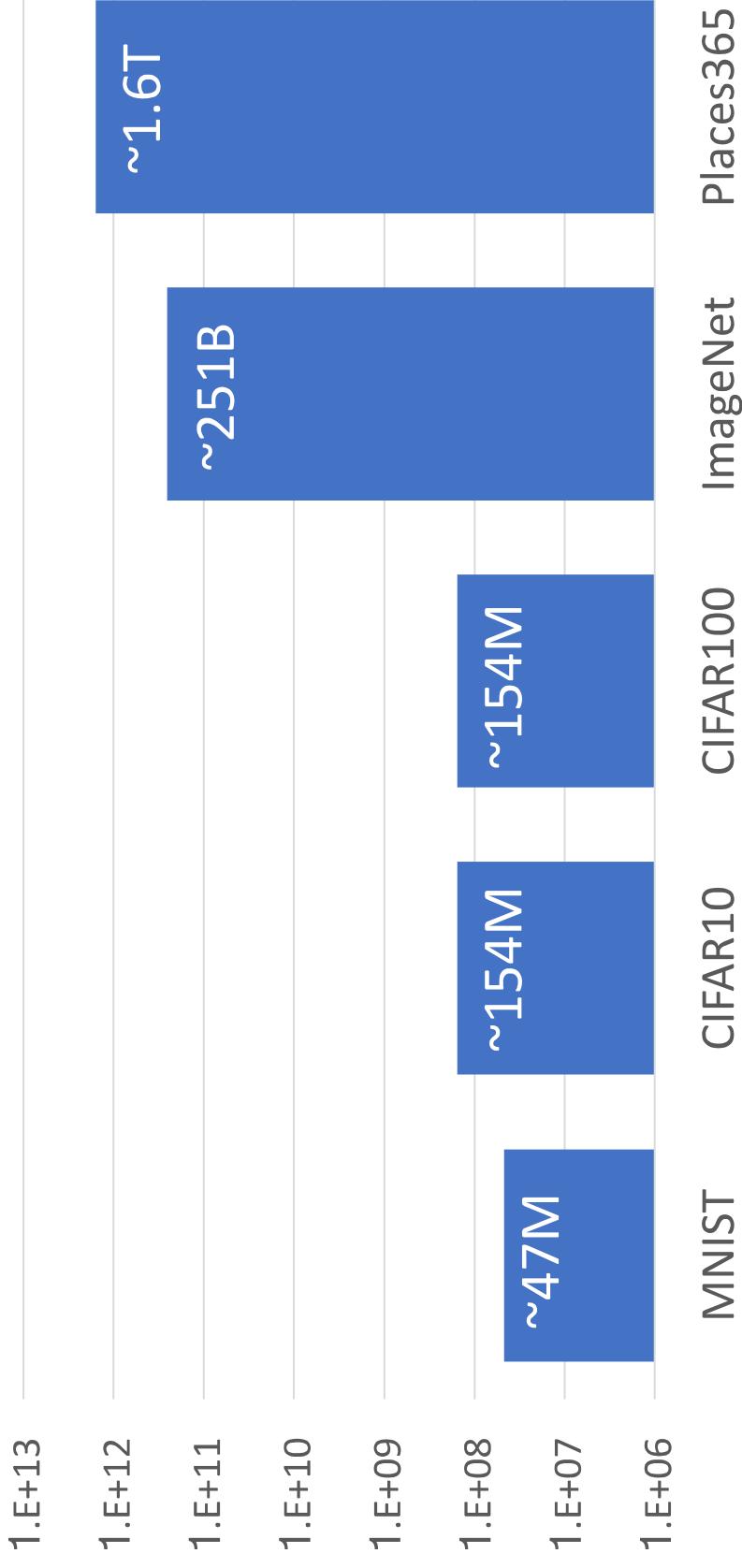


Image Classification Datasets: Omniglot



1623 categories: characters
from 50 different alphabets

20 images per category

Meant to test **few shot learning**

Lake et al, "Human-level concept learning through probabilistic program induction", Science, 2015

First classifier: Nearest Neighbor

```
def train(images, labels):  
    # Machine learning!  
    return model
```

Memorize all data
and labels

```
def predict(model, test_images):  
    # Use model to predict labels  
    return test_labels
```

Predict the label of
the most similar
training image

Distance Metric to compare images

L1 distance:

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

test image				training image				pixel-wise absolute value differences			
56	32	10	18	10	20	24	17	46	12	14	1
90	23	128	133	8	10	89	100	82	13	39	33
24	26	178	200	-	12	16	178	170	=	12	10
2	0	255	220		4	32	233	112		2	32
											108

→ 456

Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """
        X is N x D where each row is an example. Y is 1-dimension of size N
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """
        X is N x D where each row is an example we wish to predict label for
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """
        X is N x D where each row is an example. Y is 1-dimension of size N
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y
        """

    def predict(self, X):
        """
        X is N x D where each row is an example we wish to predict label for
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
        """

Memorize training data
```

Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """
        X is N x D where each row is an example. Y is 1-dimension of size N
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """
        X is N x D where each row is an example we wish to predict label for
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

For each test image:

Find nearest training image

Return label of nearest image

Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """
        X is N x D where each row is an example we wish to predict label for
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """
        X is N x D where each row is an example we wish to predict label for
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

Q: With N examples,
how fast is training?

September 9, 2019

Lecture 2 - 42

Justin Johnson

Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """
        X is N x D where each row is an example. Y is 1-dimension of size N
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """
        X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

Q: With N examples,
how fast is training?

A: O(1)

September 9, 2019

Lecture 2 - 43

Justin Johnson

Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """
        X is N x D where each row is an example. Y is 1-dimension of size N
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """
        X is N x D where each row is an example we wish to predict label for ""
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

Q: With N examples,
how fast is training?

A: O(1)

Q: With N examples,
how fast is testing?

```
# loop over all test rows
for i in xrange(num_test):
    # find the nearest training image to the i'th test image
    # using the L1 distance (sum of absolute value differences)
    distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
    min_index = np.argmin(distances) # get the index with smallest distance
    Ypred[i] = self.ytr[min_index] # predict the label of the nearest example
```

Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """
        X is N x D where each row is an example. Y is 1-dimension of size N
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """
        X is N x D where each row is an example we wish to predict label for ""
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

Q: With N examples,
how fast is training?

A: O(1)

Q: With N examples,
how fast is testing?

A: O(N)

September 9, 2019

Lecture 2 - 45

Justin Johnson

Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """
        X is N x D where each row is an example we wish to predict label for
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """
        X is N x D where each row is an example we wish to predict label for
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

Q: With N examples,
how fast is training?

A: O(1)

Q: With N examples,
how fast is testing?

A: O(N)

This is **bad**: We can afford slow training, but we need fast testing!

Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """
        X is N x D where each row is an example. Y is 1-dimension of size N
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """
        X is N x D where each row is an example we wish to predict label for
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

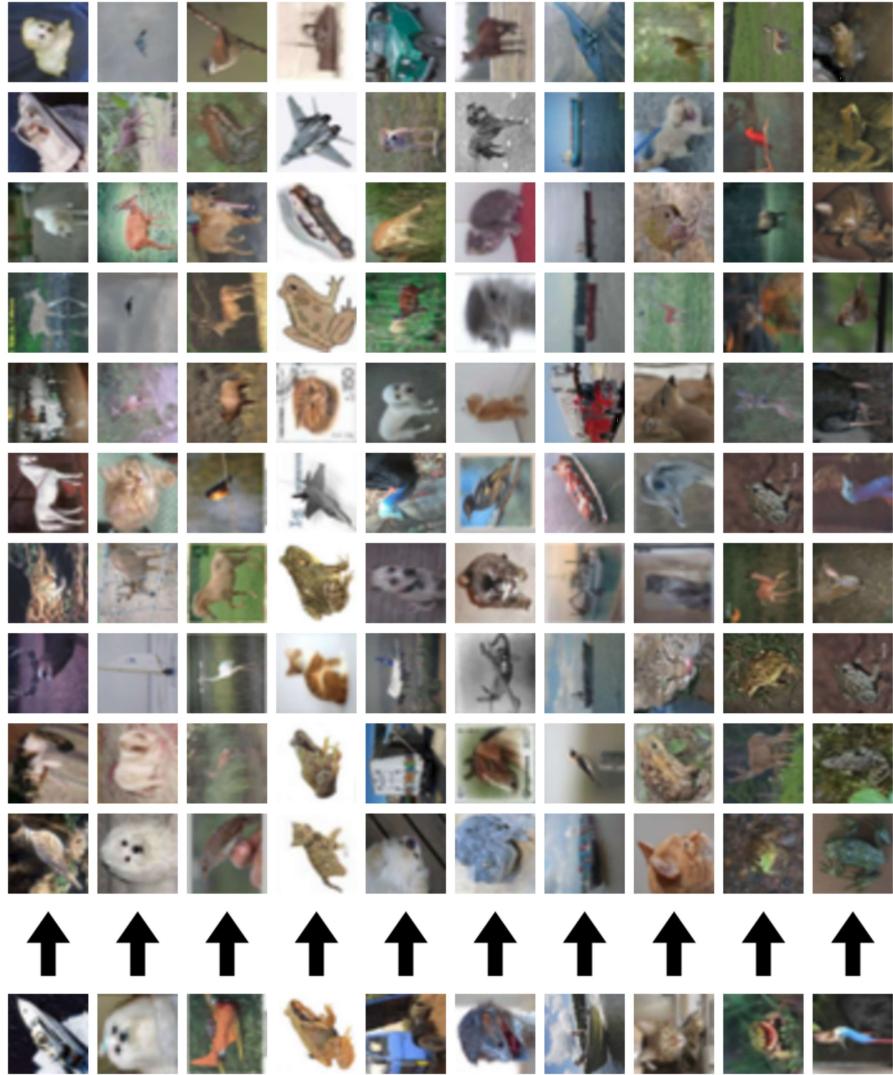
        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

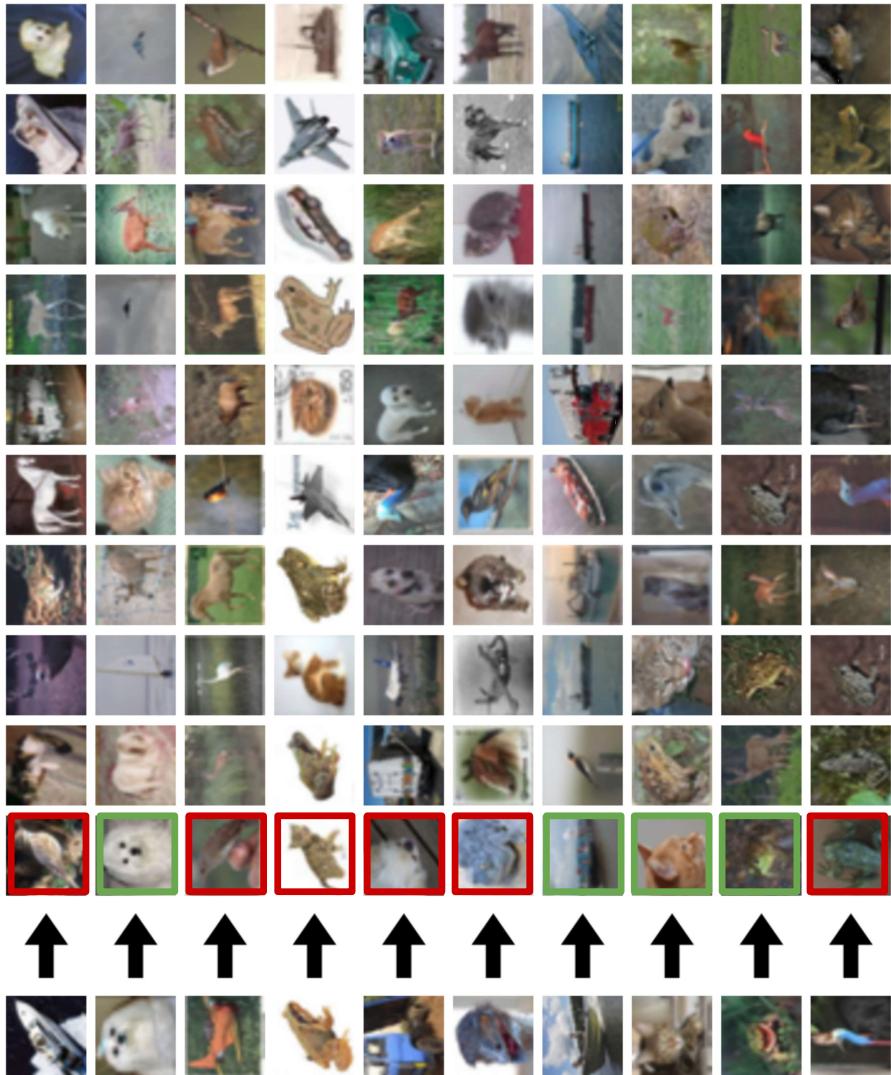
There are many methods for fast / approximate nearest neighbors; e.g. see

<https://github.com/facebookresearch/faiss>

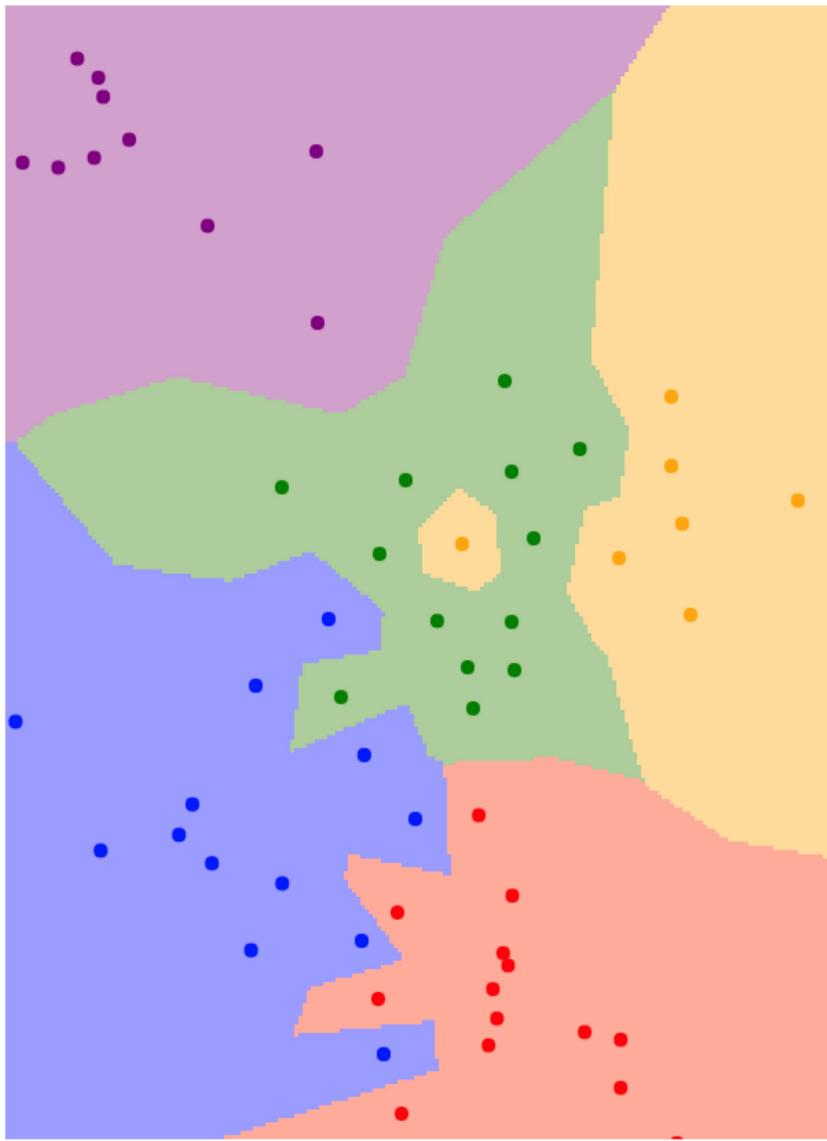
What does this look like?



What does this look like?

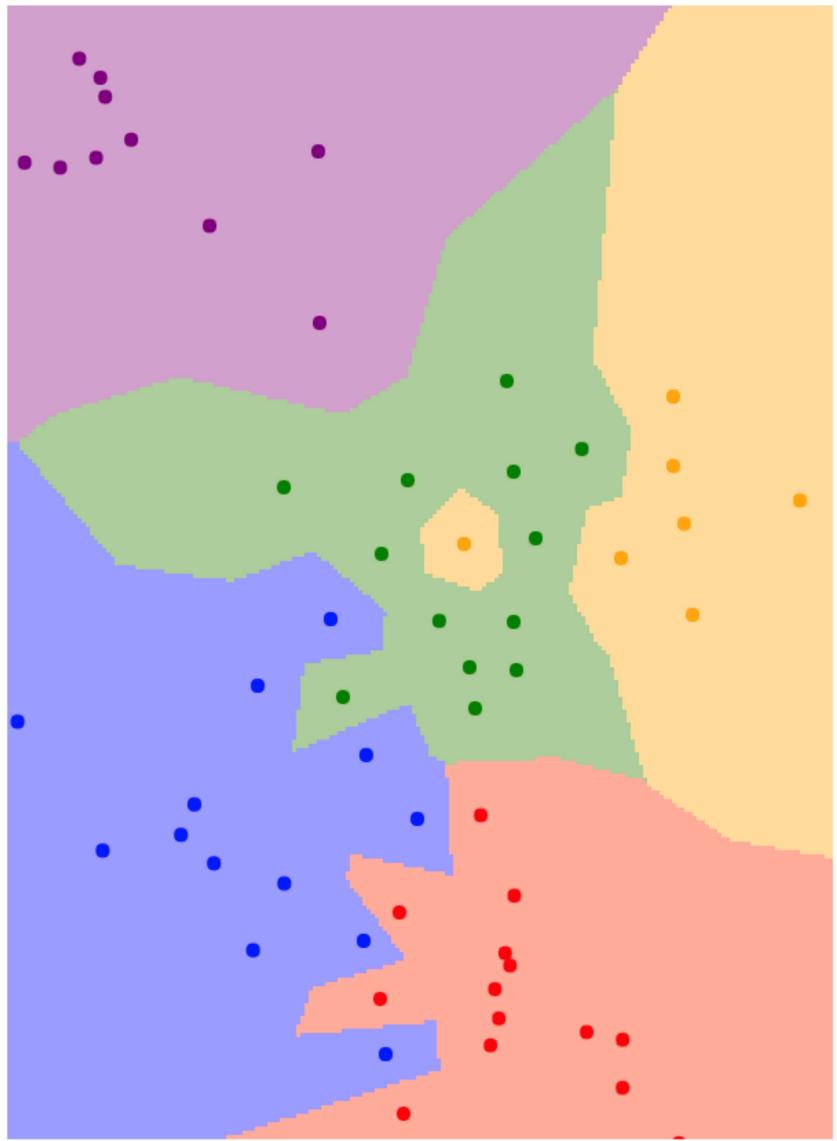


Nearst Neighbor Decision Boundaries

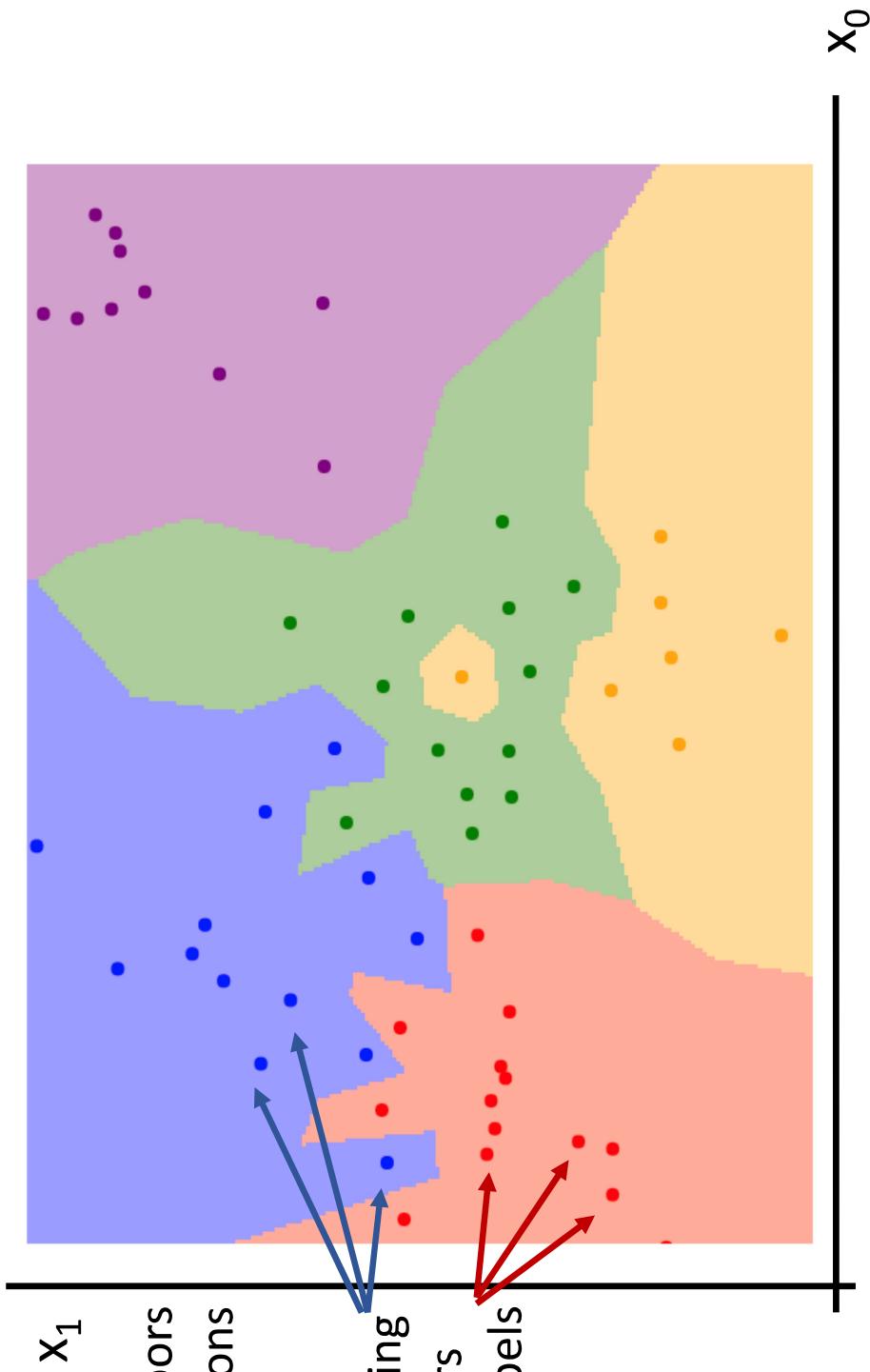


Nearest Neighbor Decision Boundaries

Nearest neighbors
in two dimensions



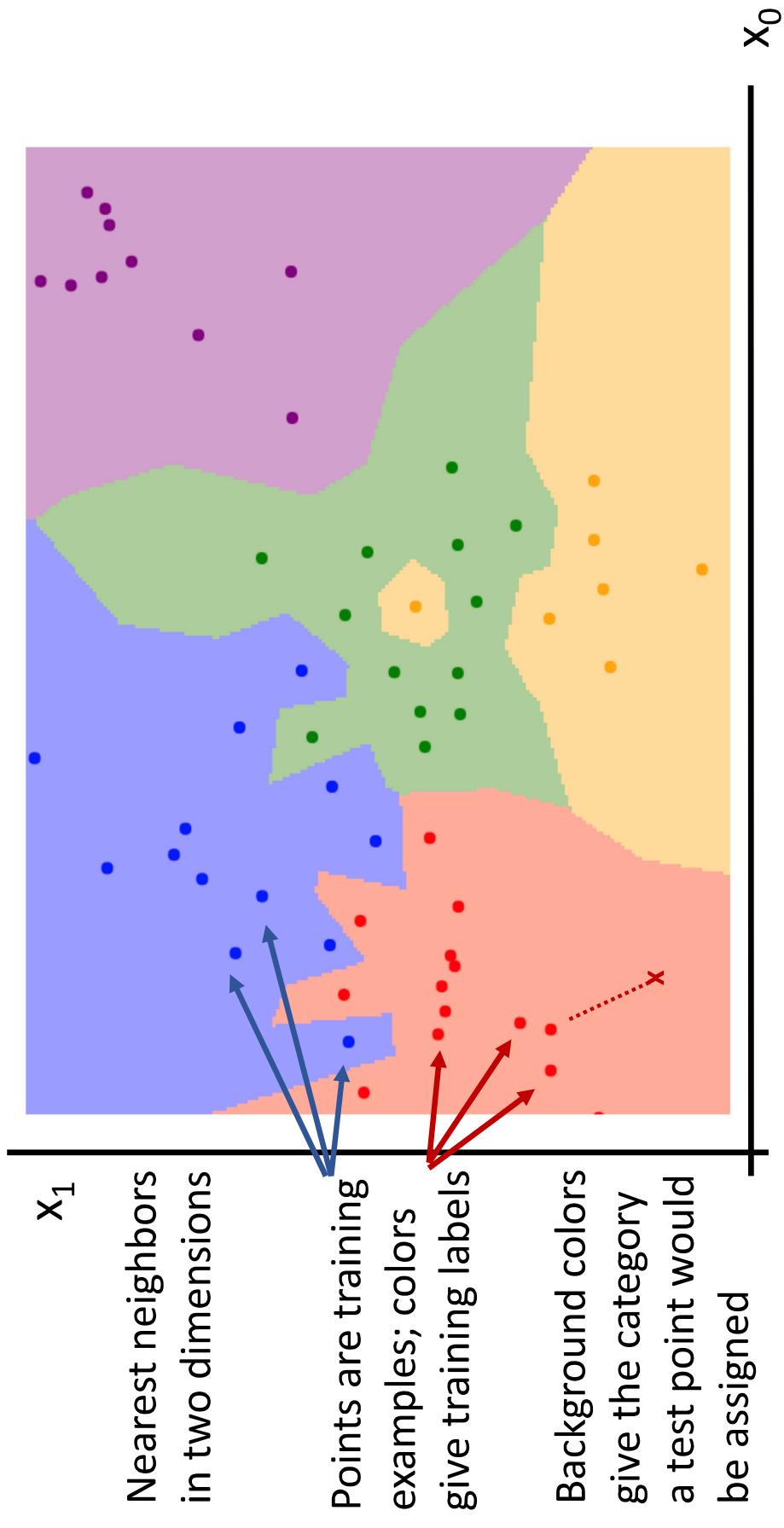
Nearest Neighbor Decision Boundaries



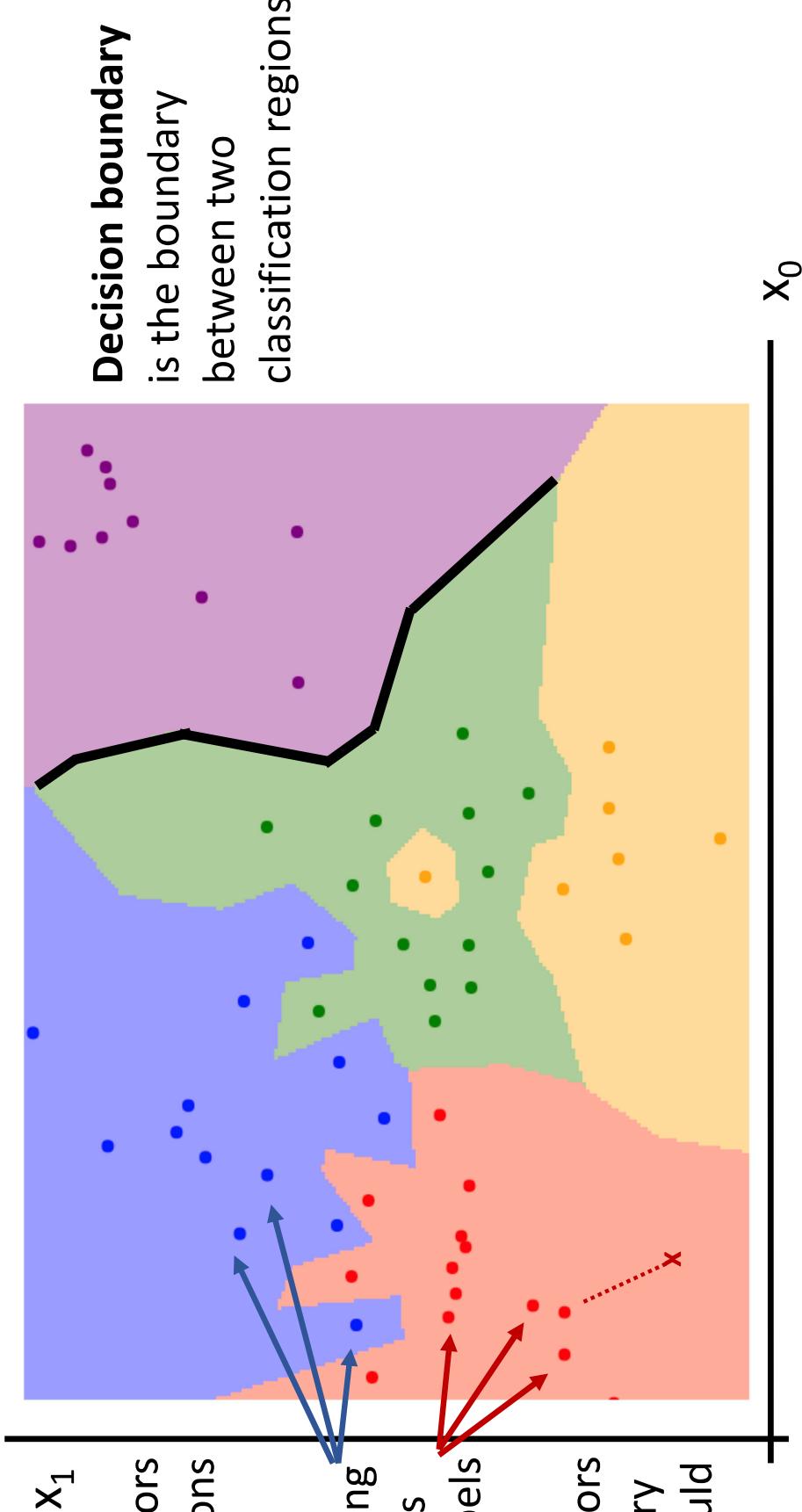
Nearest neighbors
in two dimensions

Points are training
examples; colors
give training labels

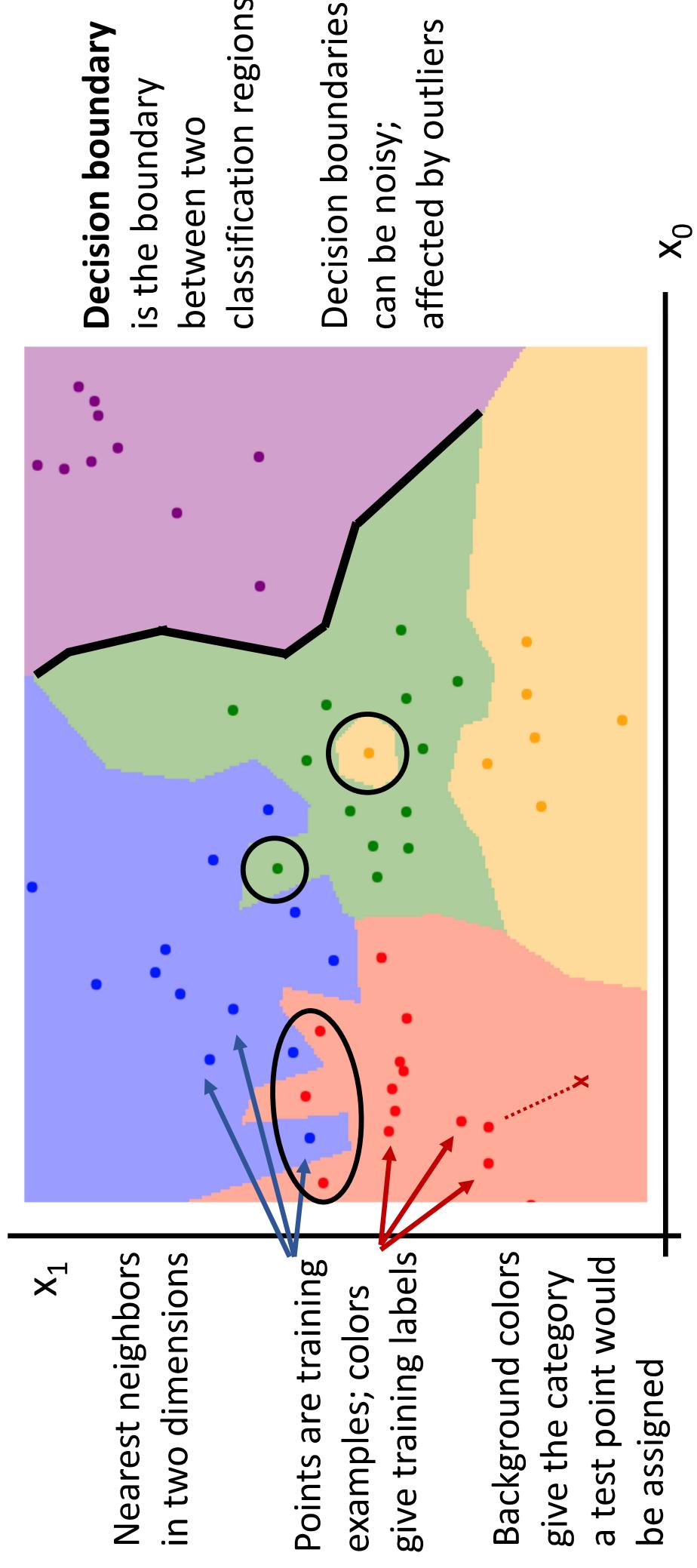
Nearest Neighbor Decision Boundaries



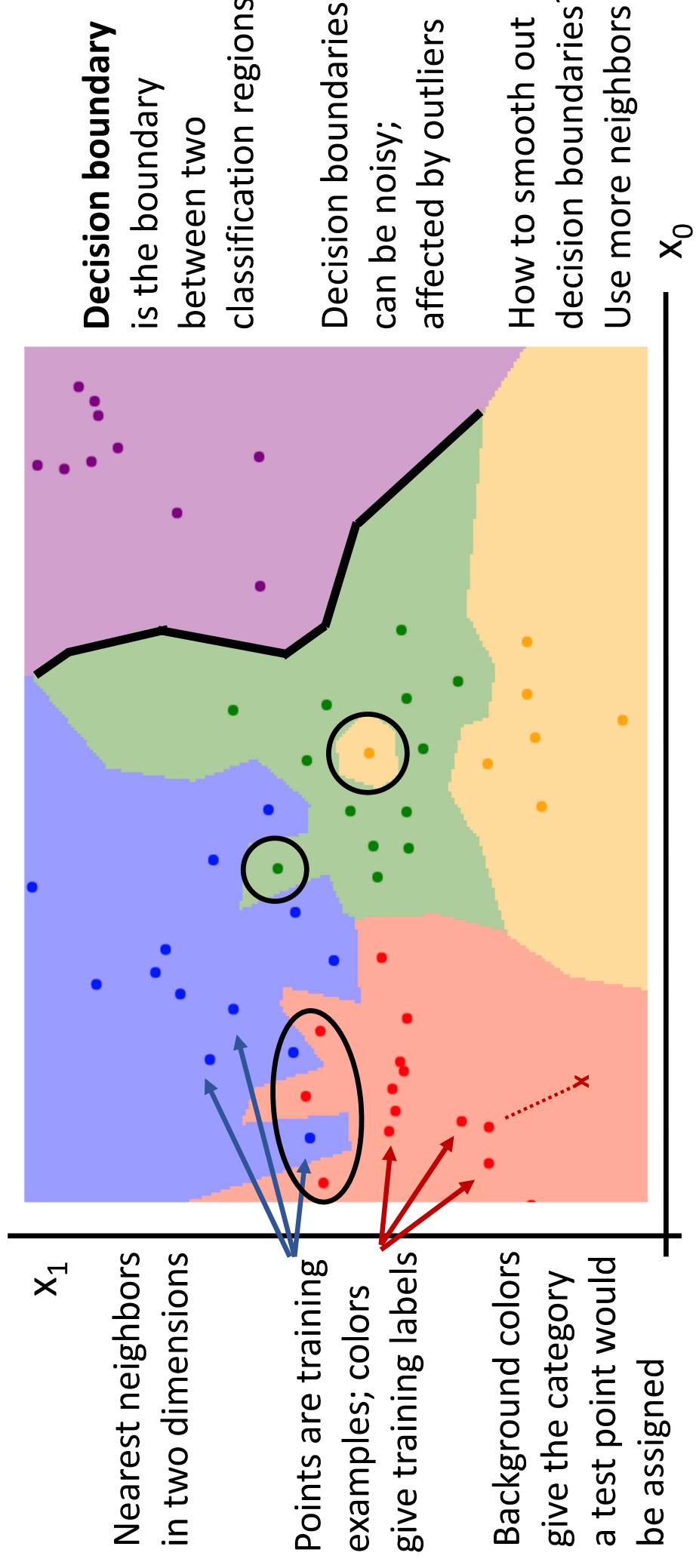
Nearest Neighbor Decision Boundaries



Nearest Neighbor Decision Boundaries



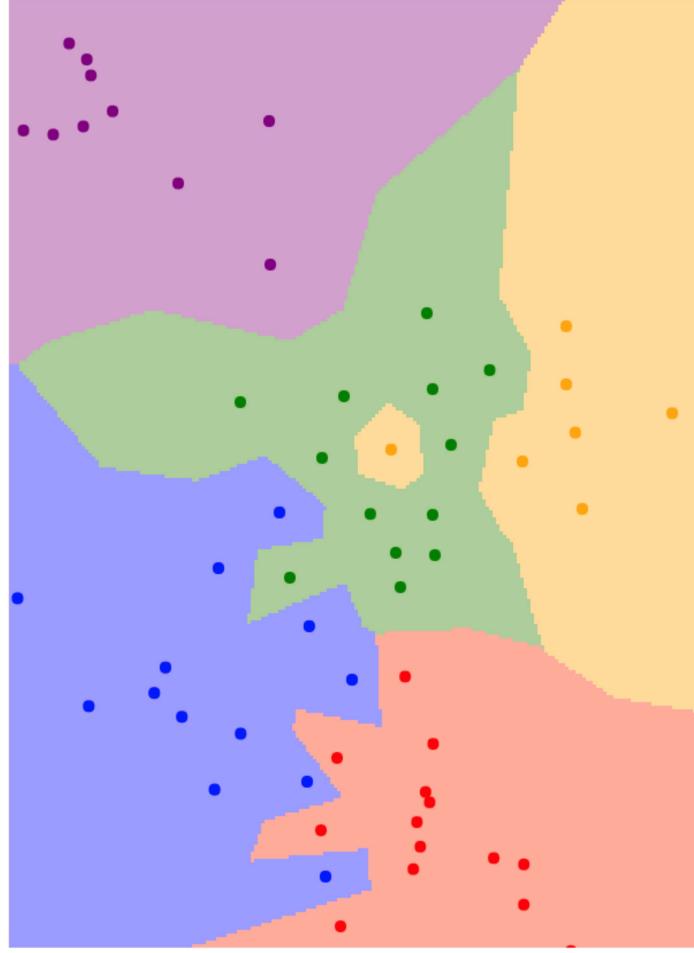
Nearest Neighbor Decision Boundaries



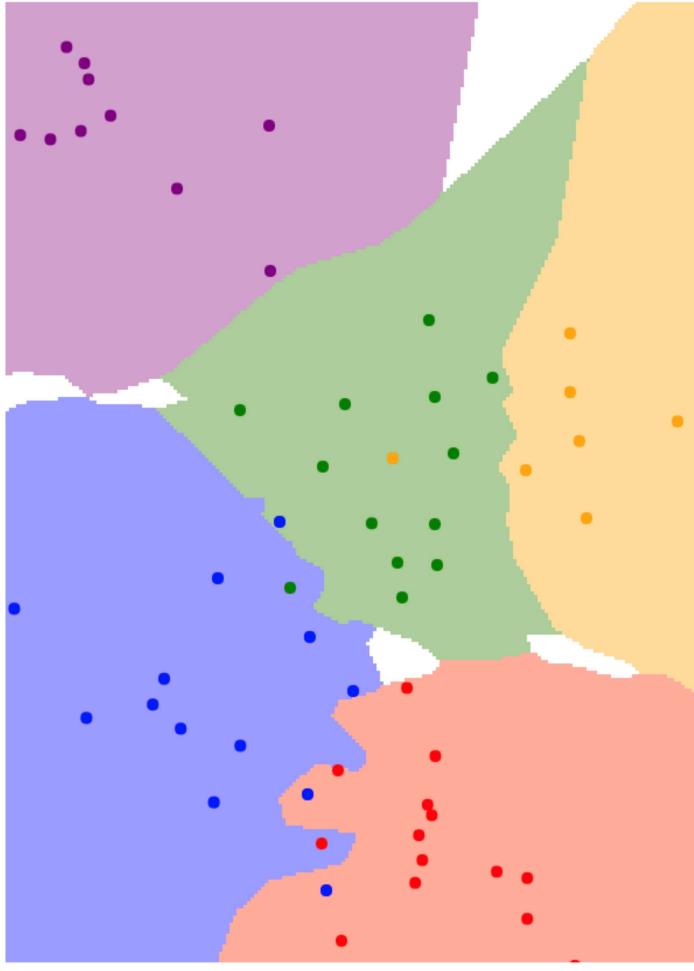
K-Nearest Neighbors

Instead of copying label from nearest neighbor,
take **majority vote** from K closest points

$K = 1$



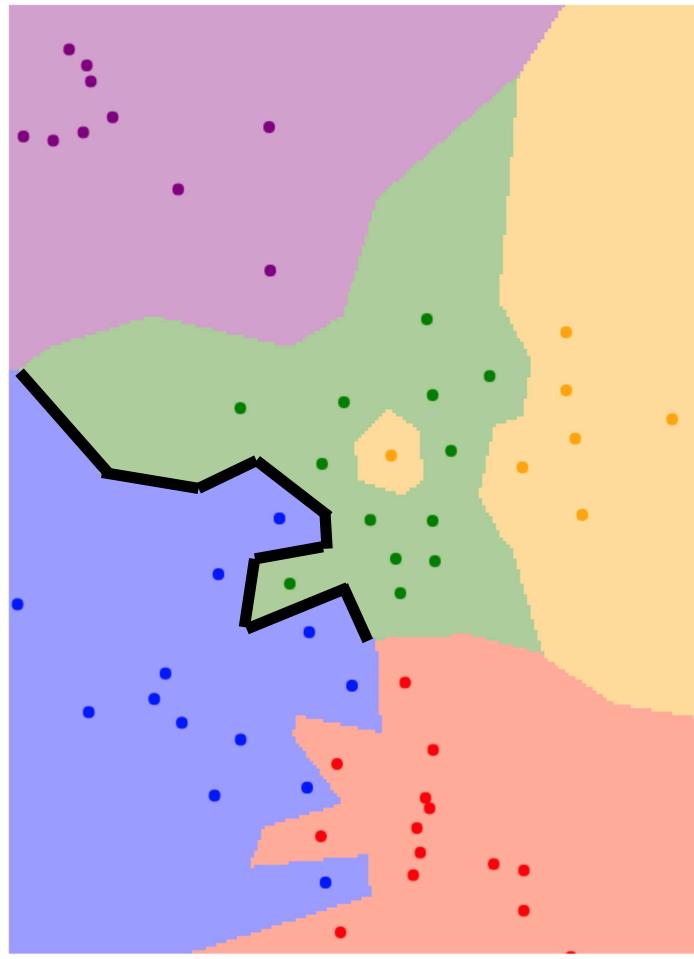
$K = 3$



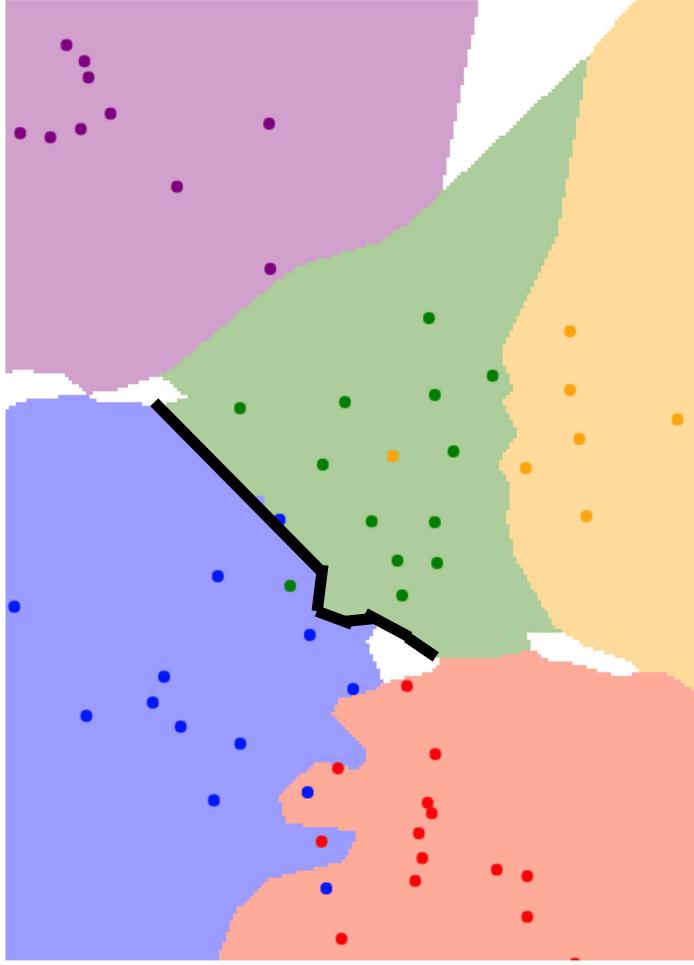
K-Nearest Neighbors

Using more neighbors helps smooth out rough decision boundaries

$K = 1$



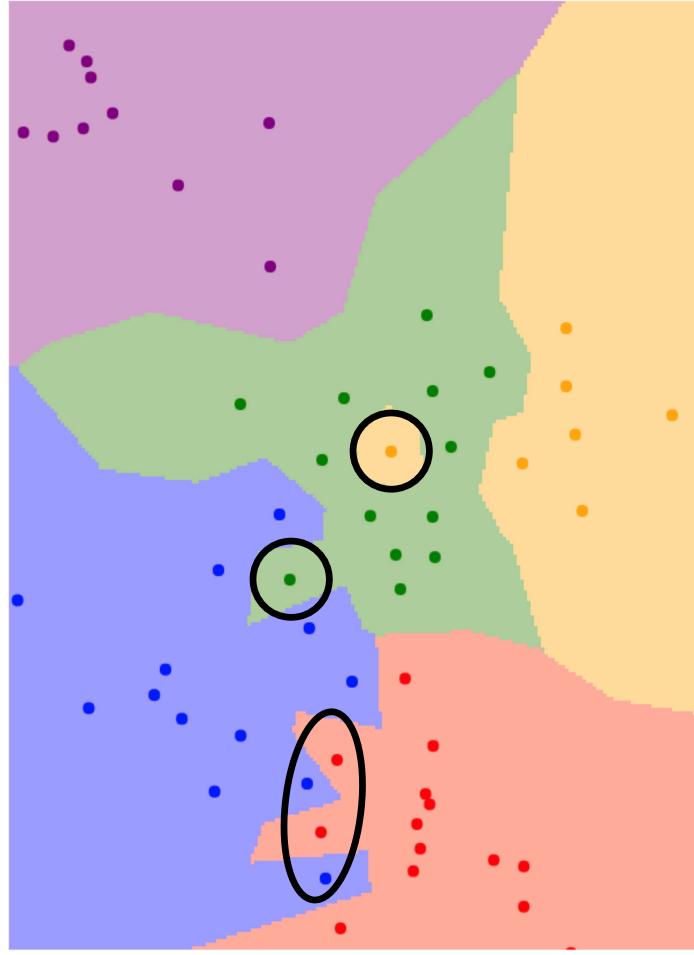
$K = 3$



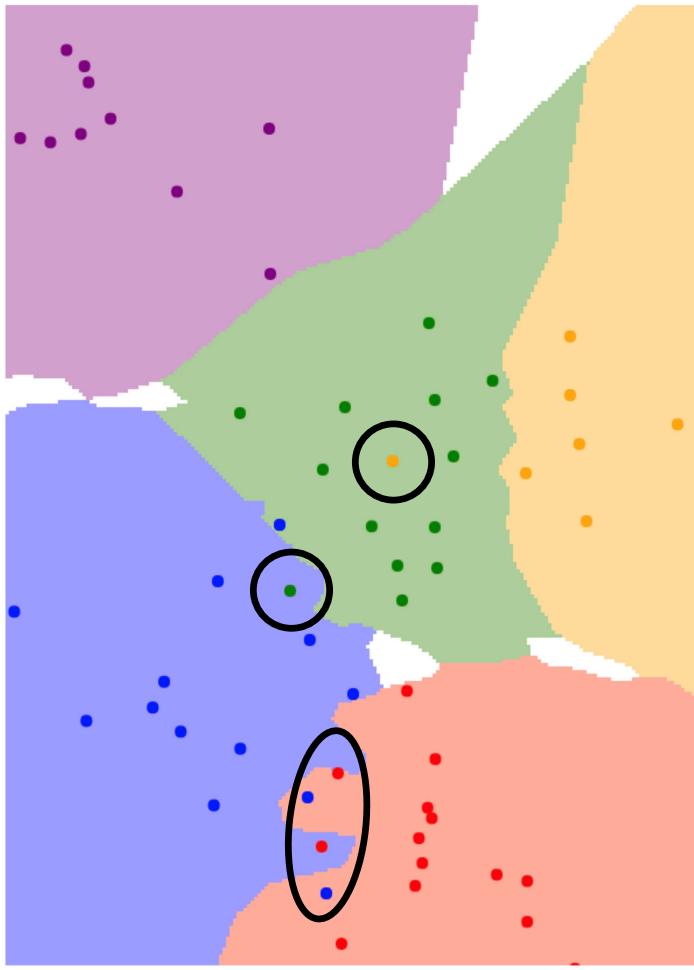
K-Nearest Neighbors

Using more neighbors helps
reduce the effect of outliers

$K = 1$



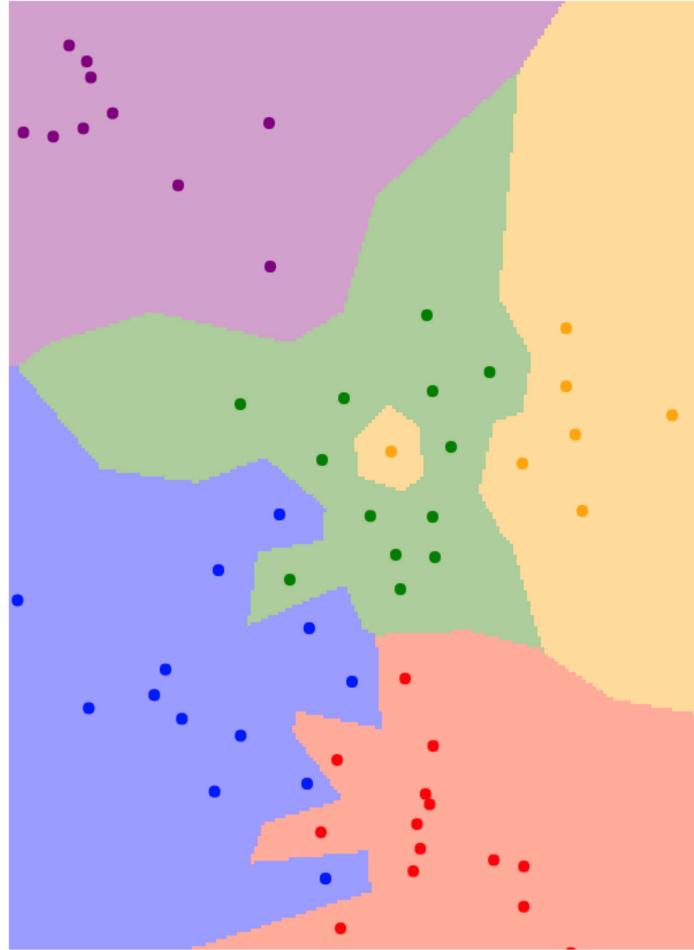
$K = 3$



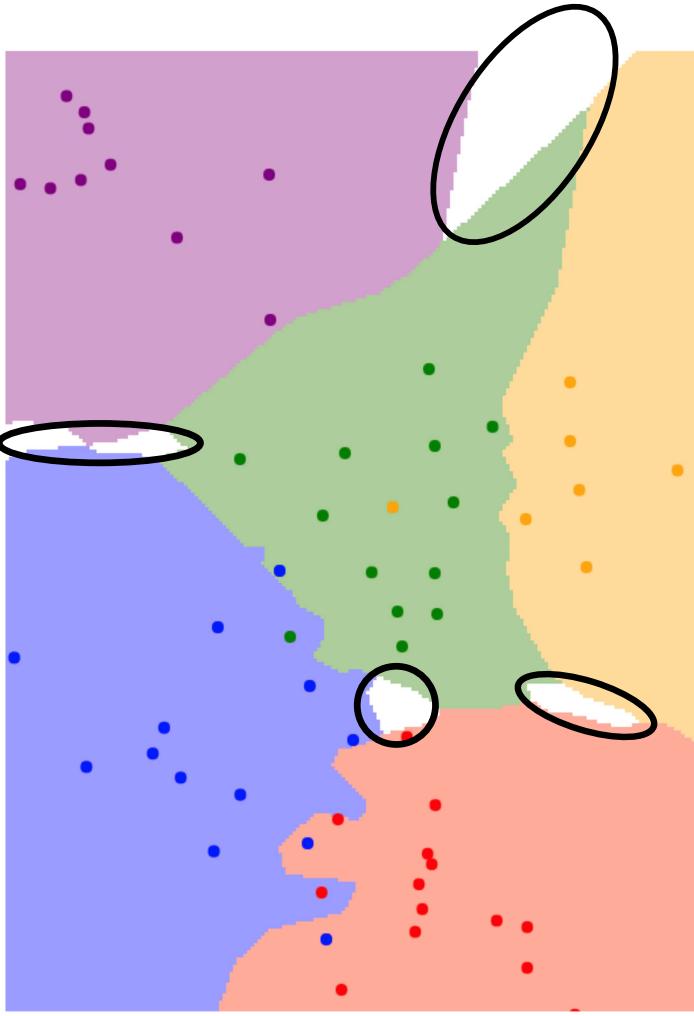
K-Nearest Neighbors

When $K > 1$ there can be
ties between classes.
Need to break somehow!

$K = 1$



$K = 3$



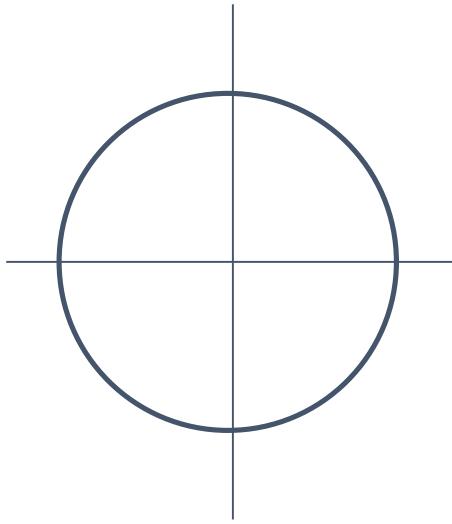
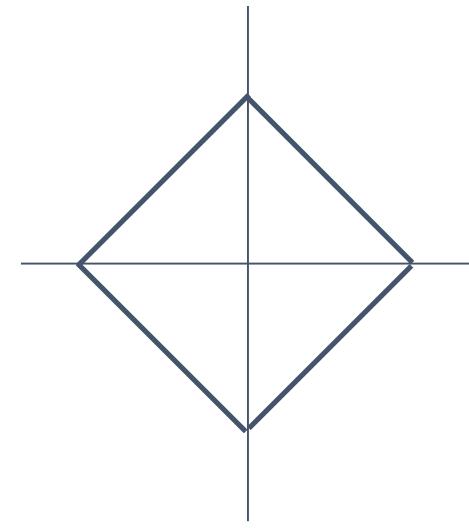
K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

L2 (Euclidean) distance

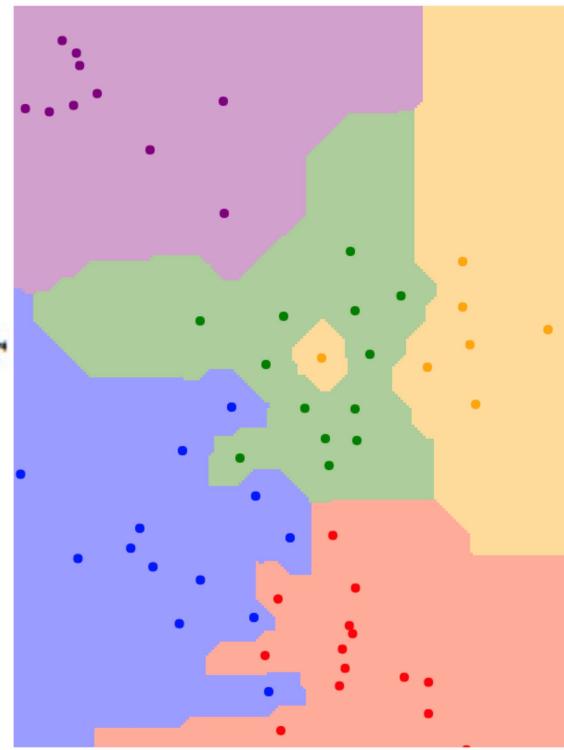
$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

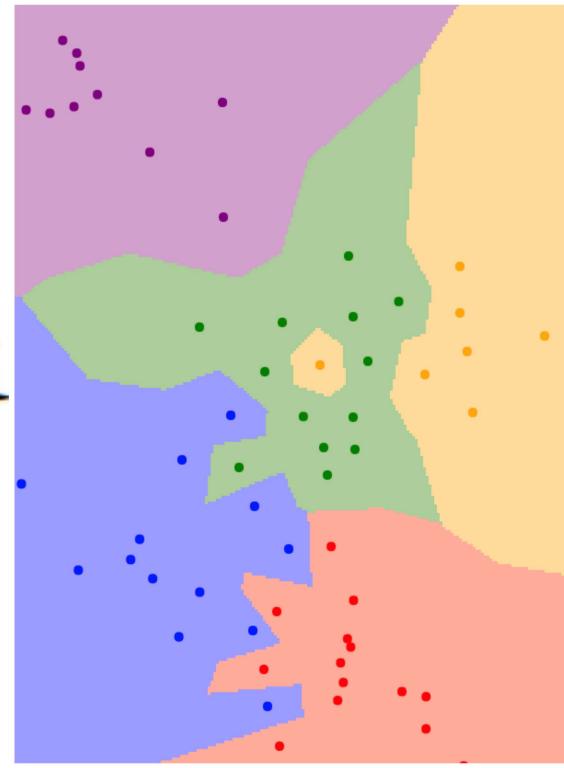
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



K = 1

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



K = 1

K-Nearest Neighbors: Distance Metric

With the right choice of distance metric, we can apply K-Nearest Neighbor to any type of data!

K-Nearest Neighbors: Distance Metric

With the right choice of distance metric, we can apply K-Nearest Neighbor to any type of data!

Mesh R-CNN
Georgia Gkioxari, Jitendra Malik, Justin Johnson
6/6/2019 cs.CV

Example:
Compare
research



papers using
tf-idf similarity

Rapid advances in 2D perception have led to systems that accurately detect objects in real-world images. However, these systems make predictions in 2D, ignoring the 3D structure of the world. Concurrently, advances in 3D shape prediction have mostly focused on synthetic benchmarks and isolated objects. We unify advances in these two areas. We propose a system that detects objects in real-world images and produces a triangle mesh giving the full 3D shape of each detected object. Our system, called Mesh R-CNN, augments Mask R-CNN with a mesh prediction branch that outputs meshes with varying topological structure by first predicting coarse voxel representations which are converted to meshes and refined with a graph convolution network operating over the mesh's vertices and edges. We validate our mesh prediction branch on ShapeNet, where we outperform prior work on single-image shape prediction. We then deploy our full Mesh R-CNN system on Pix3D, where we jointly detect objects and predict their 3D shapes.

<http://www.arxiv-sanity.com/search?q=mesh+r-cnn>

K-Nearest Neighbors: Distance Metric

Most similar papers:

[Pixel2Mesh++: Multi-View 3D Mesh Generation via Deformation](#)
Chao Wen, Yinda Zhang, Zhiwen Li, Yanwei Fu
8/16/2019 (v1: 8/5/2019) cs.CV
Accepted by ICCV 2019

1906.06543v2 pdf

show similar | discuss

[Image-based 3D Object Reconstruction: State-of-the-Art and Trends in the Deep Learning Era](#)
Xian-Feng Lan, Hamid Laqa, Mohammed Bennamoun
6/18/2019 (v1: 6/15/2019) cs.CV | cs.CG | cs.GR | cs.CV
Accepted by CVPR 2019

1906.06543v2 pdf

show similar | discuss

[3D reconstruction is a longstanding ill-posed problem, which has been explored for decades by the computer vision, computer graphics, and machine learning communities. Since 2015, image-based 3D reconstruction using convolutional neural networks \(CNN\) has attracted increasing interest and demonstrated an impressive performance. Given this new era of rapid evolution, this article provides a comprehensive survey of the recent developments in this field. We focus on the works which use deep learning techniques to estimate the 3D shape of generic objects either from a single or multiple RGB images. We organize the literature based on the shape representations, the network architectures, and the training mechanisms they use. While this survey is intended for methods which reconstruct generic objects, we also review some of the recent works which focus on specific object classes such as human body shapes and faces. We provide an analysis and comparison of the performance of some key papers, summarize some of the open problems in this field, and discuss promising directions for future research.](#)

[Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images](#)
Nanyang Wang, Yinda Zhang, Zhiwen Li, Yanwei Fu, Wei Liu, Yu-Gang Jiang
8/3/2018 (v1: 4/5/2018) cs.CV
Accepted by CVPR 2018

1804.01654v2 pdf

show similar | discuss

[We propose an end-to-end deep learning architecture that produces a 3D shape in triangular mesh from a single color image. Limited by the nature of deep neural network, previous methods usually represent a 3D shape in volume or point cloud, and it is non-trivial to convert them to the more ready-to-use mesh model. Unlike the existing methods, our network represents 3D mesh in a graph-based convolutional neural network and produces correct geometry by progressively deforming an ellipsoid, leveraging perceptual features extracted from the input image. We adopt a coarse-to-fine strategy to make the whole deformation procedure stable, and define various of mesh related losses to capture properties of different levels to guarantee visually appealing and physically accurate 3D geometry. Extensive experiments show that our method not only qualitatively produces mesh model with better details, but also achieves higher 3D shape estimation accuracy compared to the state-of-the-art.](#)

[Pixel2Mesh++: Multi-View 3D Mesh Generation via Deformation](#)
Chao Wen, Yinda Zhang, Zhiwen Li, Yanwei Fu
8/16/2019 (v1: 8/5/2019) cs.CV
Accepted by ICCV 2019

1906.06543v2 pdf

show similar | discuss

[We study the problem of shape generation in 3D mesh representation from few color images with known camera poses. While many previous works learn to hallucinate the shape directly from priors, we resort to further improving the shape quality by leveraging cross-view information with a graph convolutional network. Instead of building a direct mapping function from images to 3D shape, our model learns to predict series of deformations to improve a coarse shape iteratively. Inspired by traditional multiple view geometry methods, our network samples nearby area around the initial mesh's vertex locations and reasons an optimal deformation using perceptual feature statistics built from multiple input images. Extensive experiments show that our model produces accurate 3D shape that are not only visually plausible from the input perspectives, but also well aligned to arbitrary viewpoints. With the help of physically driven architecture, our model also exhibits generalization capability across different semantic categories, number of input images, and quality of mesh initialization.](#)

[GEOMetrics: Exploiting Geometric Structure for Graph-Encoded Objects](#)
Edward J. Smith, Scott Fujimoto, Adriana Romero, David Meger
1/31/2019 cs.CV
18 pages

1901.11461v1 pdf

show similar | discuss

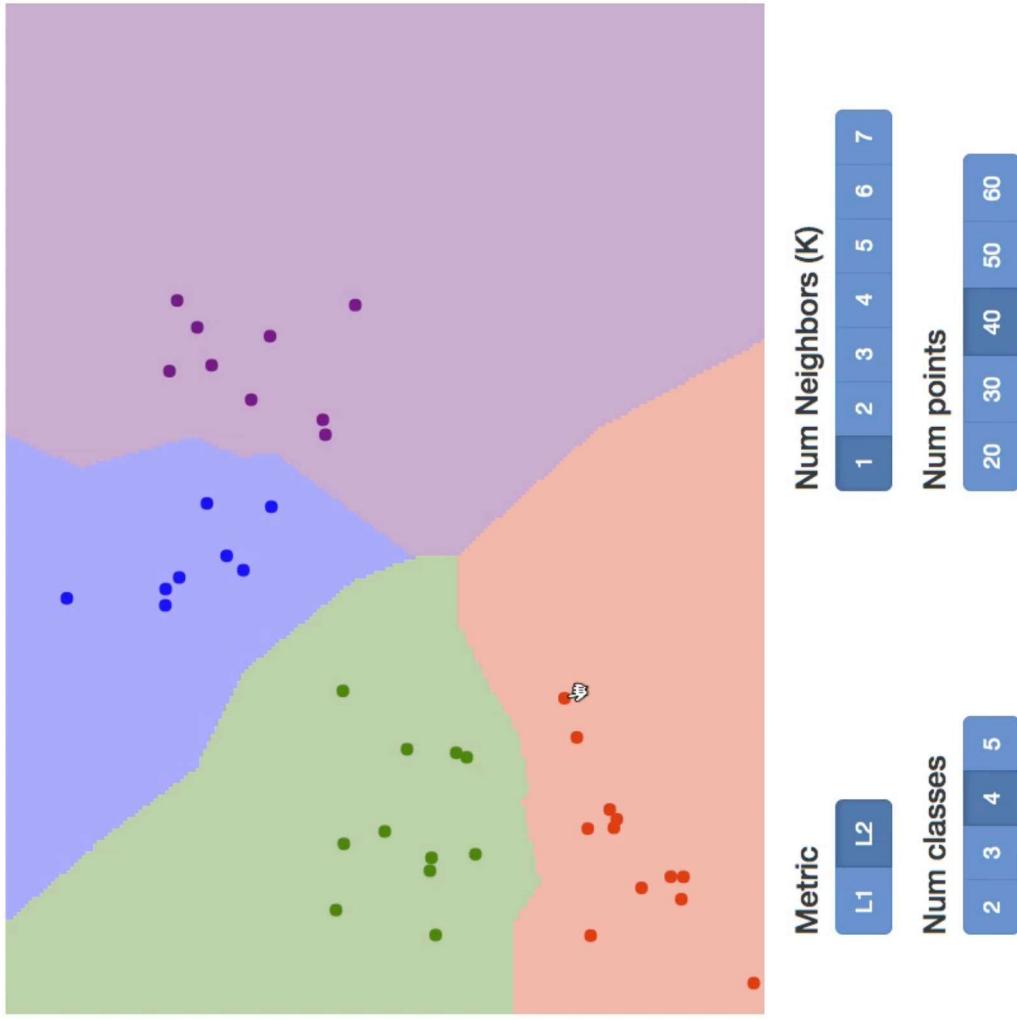
[Mesh models are a promising approach for encoding the structure of 3D objects. Current mesh reconstruction systems predict uniformly distributed vertex locations of a predetermined graph through a series of graph convolutions, leading to compromises with respect to performance or resolution. In this paper, we argue that the graph representation of geometric objects allows for additional structure, which should be leveraged for enhanced reconstruction. Thus, we propose a system which partly benefits from the advantages of the geometric structure of graph encoded objects by introducing \(1\) a graph convolutional update preserving vertex information; \(2\) an adaptive splitting heuristic allowing detail to emerge; and \(3\) a training objective operating both on the local surfaces defined by vertices as well as the global structure defined by the mesh. Our proposed method is evaluated on the task of 3D object reconstruction from images with the ShapeNet dataset, where we demonstrate state of the art performance, both visually and numerically, while having far smaller space requirements by generating adaptive meshes](#)

K-Nearest Neighbors: Web Demo

Interactively move points around
and see decision boundaries change

Play with L1 vs L2 metrics

Play with changing number of
training points, value of K



<http://vision.stanford.edu/teaching/cs231n-demos/knn/>

Hyperparameters

- What is the best value of K to use?
- What is the best **distance metric** to use?

These are examples of **hyperparameters**: choices about our learning algorithm that we don't learn from the training data; instead we set them at the start of the learning process

Hyperparameters

- What is the best value of K to use?
- What is the best **distance metric** to use?

These are examples of **hyperparameters**: choices about our learning algorithm that we don't learn from the training data; instead we set them at the start of the learning process

Very problem-dependent. In general need to try them all and see what works best for our data / task.

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

Your Dataset

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: $K = 1$ always works perfectly on training data

Your Dataset

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data



BAD: $K = 1$ always works perfectly on training data

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data



Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data



BAD: $K = 1$ always works perfectly on training data

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data



BAD: No idea how algorithm will perform on new data

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data



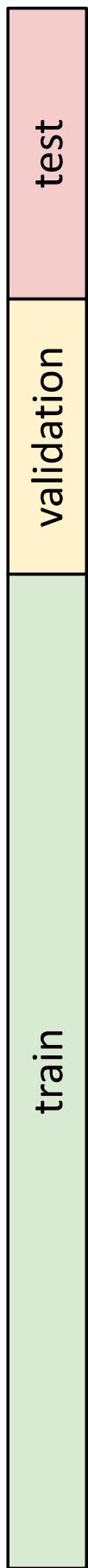
BAD: $K = 1$ always works perfectly on training data

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data



BAD: No idea how algorithm will perform on new data

Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test



Better!

Setting Hyperparameters

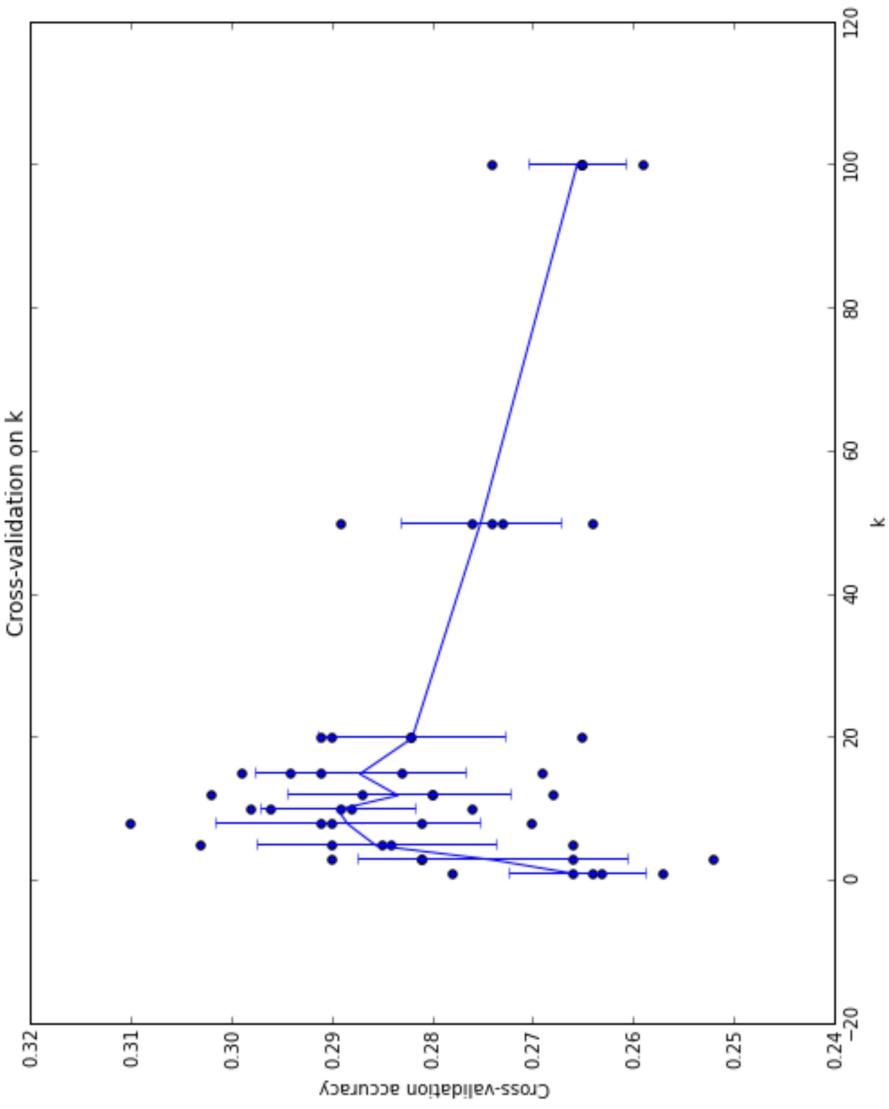
Your Dataset

Idea #4: Cross-Validation: Split data into **folds**, try each fold as validation and average the results

	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

Useful for small datasets, but (unfortunately) not used too frequently in deep learning

Setting Hyperparameters



Example of 5-fold cross-validation for the value of k .

Each point: single outcome.

The line goes through the mean, bars indicated standard deviation

(Seems that $k \sim 7$ works best for this data)

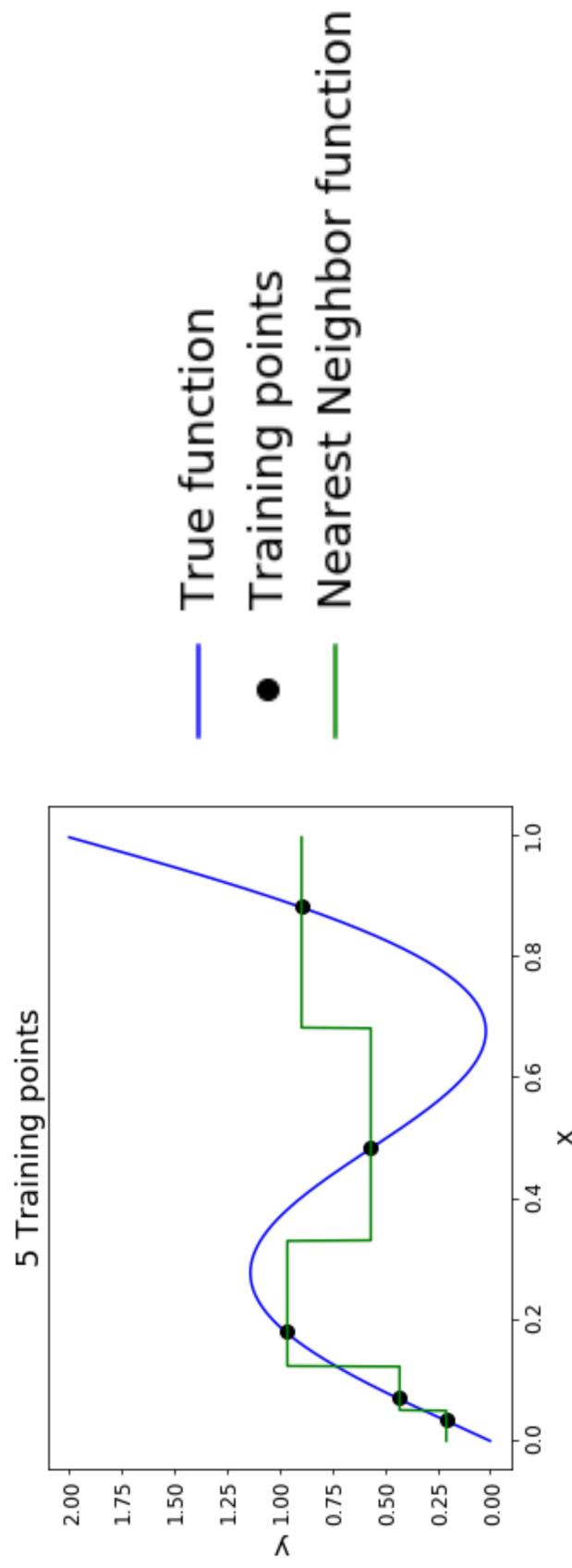
K-Nearest Neighbor: Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any^(*) function!

(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

K-Nearest Neighbor: Universal Approximation

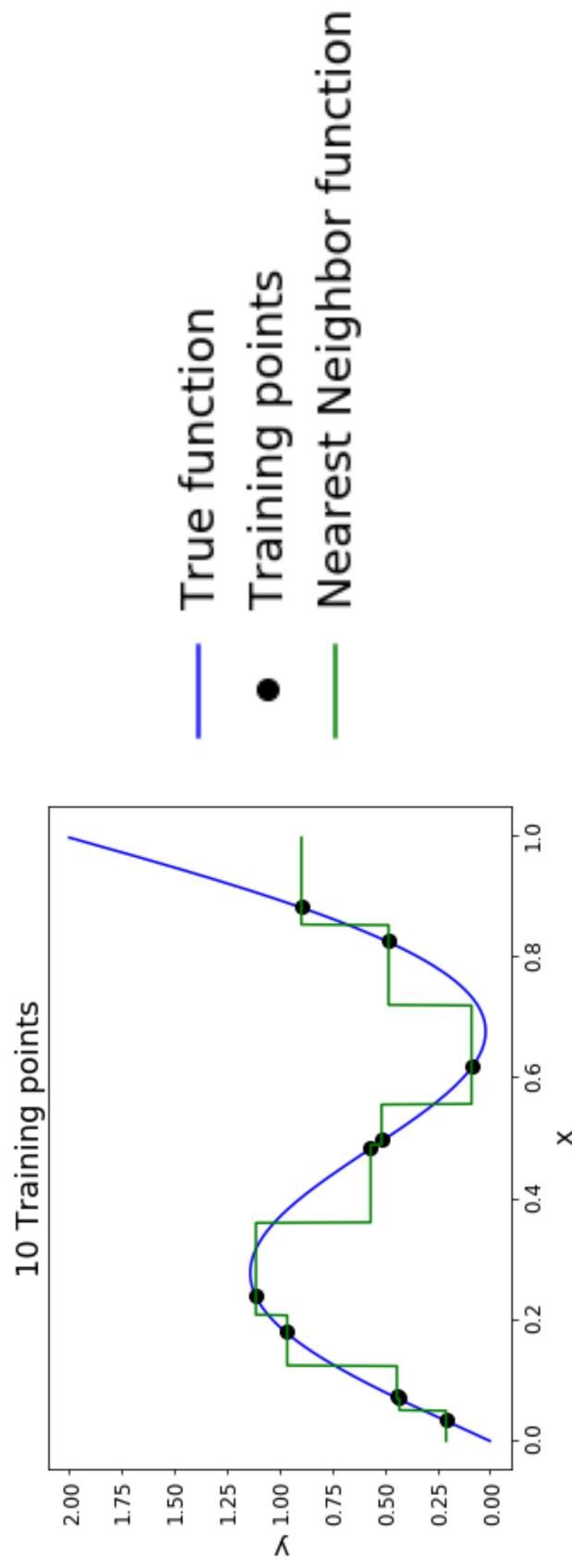
As the number of training samples goes to infinity, nearest neighbor can represent any^(*) function!



(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

K-Nearest Neighbor: Universal Approximation

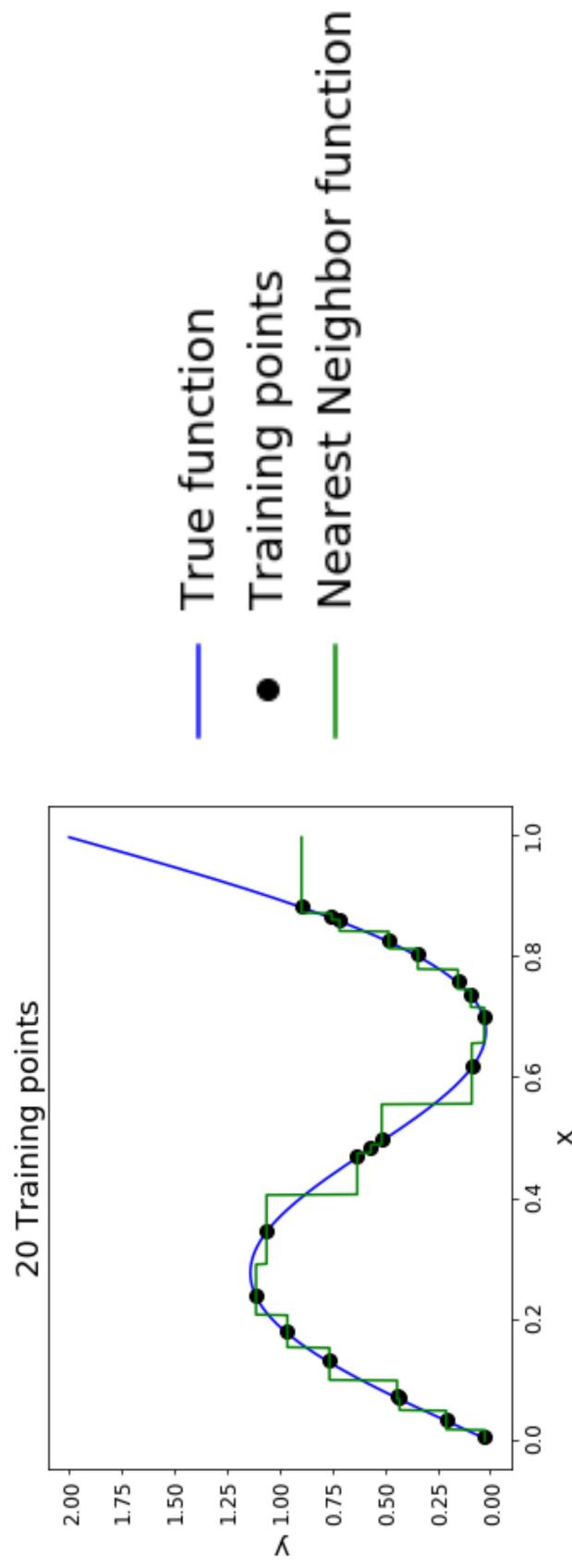
As the number of training samples goes to infinity, nearest neighbor can represent any^(*) function!



(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

K-Nearest Neighbor: Universal Approximation

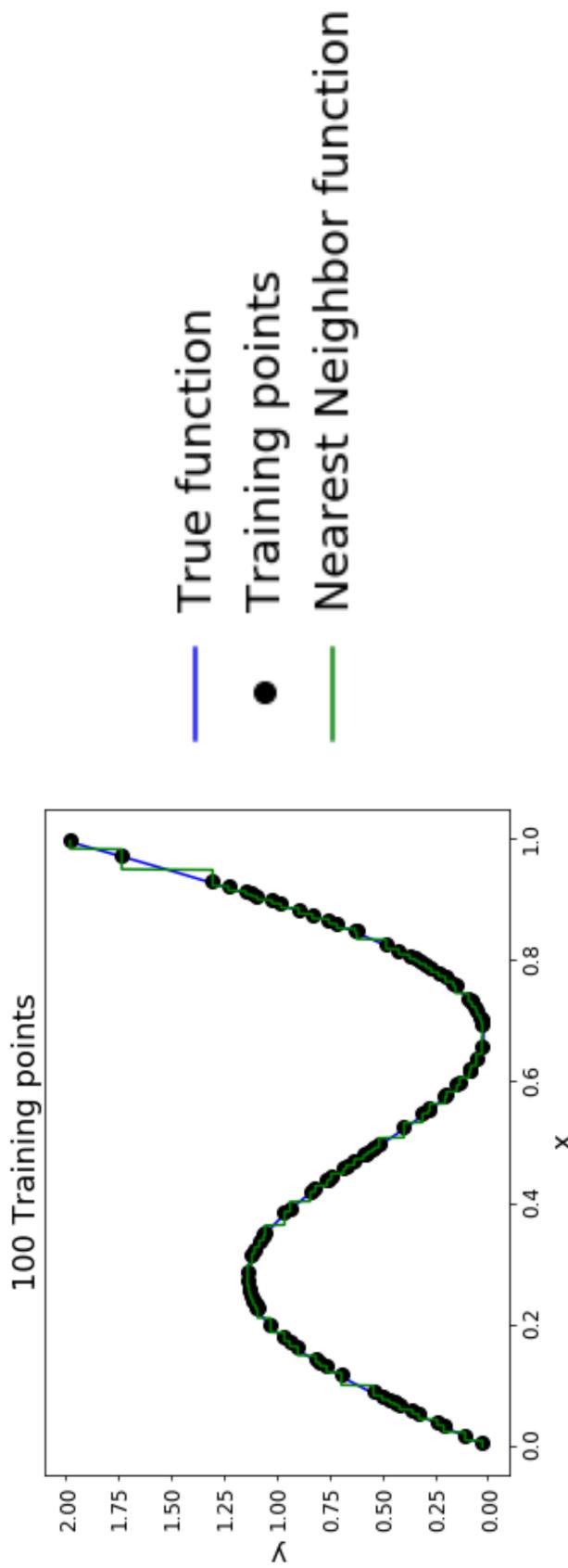
As the number of training samples goes to infinity, nearest neighbor can represent any^(*) function!



(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

K-Nearest Neighbor: Universal Approximation

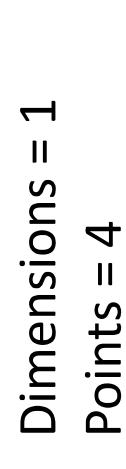
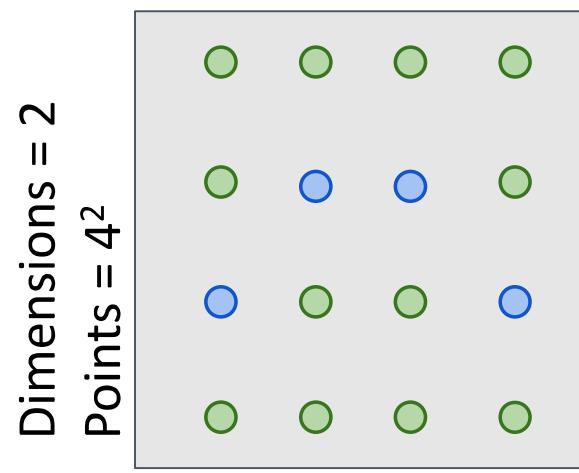
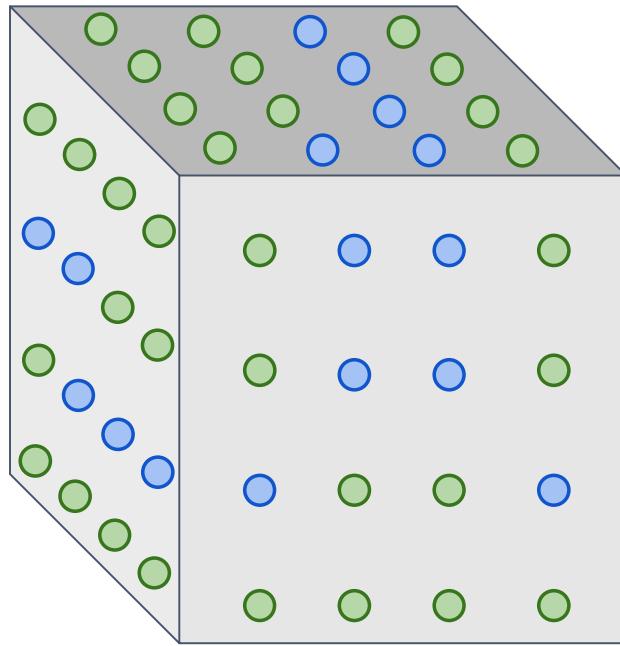
As the number of training samples goes to infinity, nearest neighbor can represent any^(*) function!



(*) Subject to many technical conditions. Only continuous functions on a compact domain; need to make assumptions about spacing of training points; etc.

Problem: Curse of Dimensionality

Curse of dimensionality: For uniform coverage of space, number of training points needed grows exponentially with dimension



Problem: Curse of Dimensionality

Curse of dimensionality: For uniform coverage of space, number of training points needed grows exponentially with dimension

Number of possible
32x32 binary images:

$$2^{32 \times 32} \approx 10^{308}$$

Problem: Curse of Dimensionality

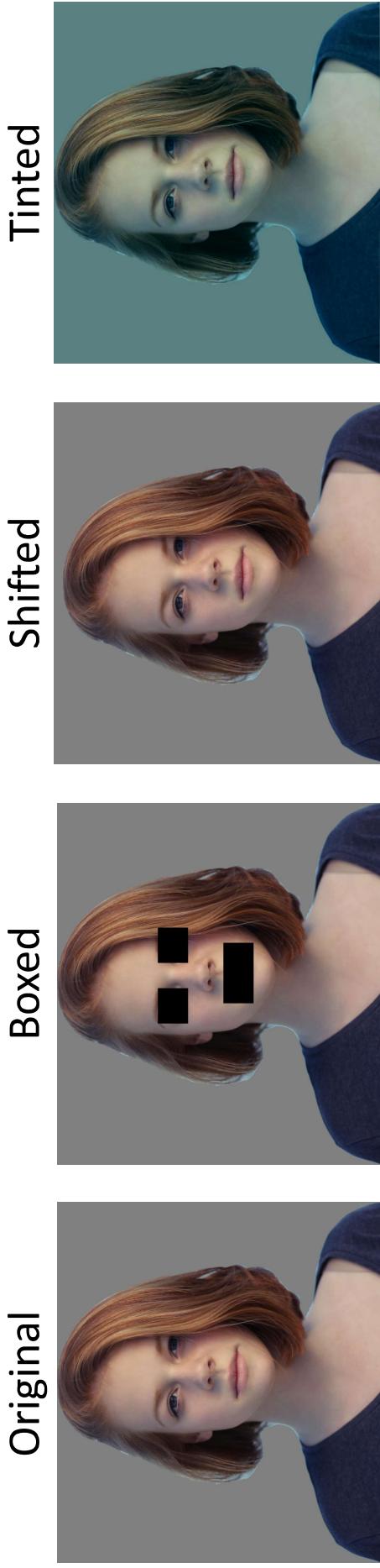
Curse of dimensionality: For uniform coverage of space, number of training points needed grows exponentially with dimension

Number of possible
32x32 binary images:
Number of elementary particles
in the visible universe: [\(source\)](#)

$$2^{32 \times 32} \approx 10^{308} \approx 10^{97}$$

K-Nearest Neighbor on raw pixels is seldom used

- Very slow at test time
- Distance metrics on pixels are not informative



(all 3 images have same L2 distance to the one on the left)

Nearest Neighbor with ConvNet features works well!



Devlin et al, "Exploring Nearest Neighbor Approaches for Image Captioning", 2015

Justin Johnson

Lecture 2 - 85

September 9, 2019

Nearest Neighbor with ConvNet features works well!

Example: Image Captioning with Nearest Neighbor



A bedroom with a
bed and a couch.



A cat sitting in a
bathroom sink.



A train is stopped at
a train station.



A wooden bench in
front of a building.

Summary

In **Image classification** we start with a **training set** of images and labels, and must predict labels on the **test set**

Image classification is challenging due to the semantic gap: we need invariance to occlusion, deformation, lighting, intraclass variation, etc

Image classification is a **building block** for other vision tasks

The **K-Nearest Neighbors** classifier predicts labels based on nearest training examples

Distance metric and K are **hyperparameters**

Choose hyperparameters using the **validation set**; only run on the test set once at the very end!

Next time: Linear Classifiers

