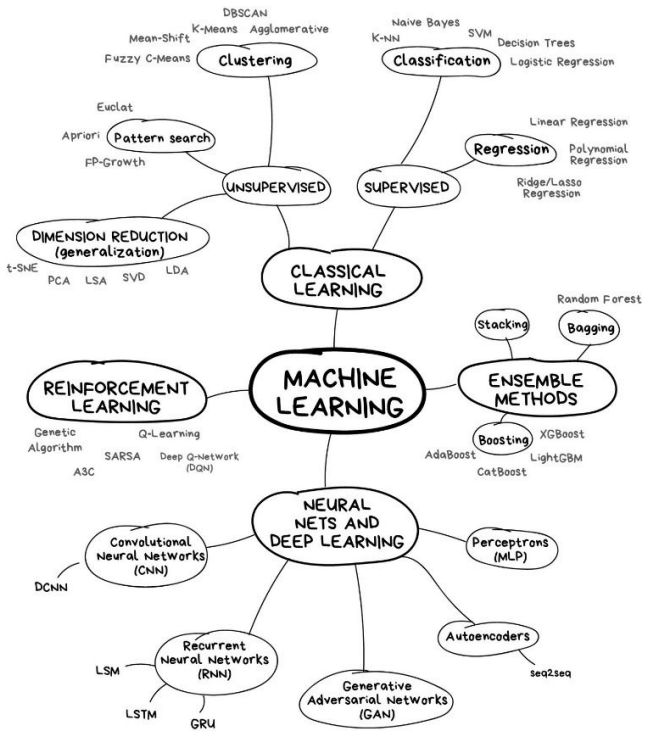
Building a Robot Judge: Data Science for Decision-Making

6. Deep Learning Essentials

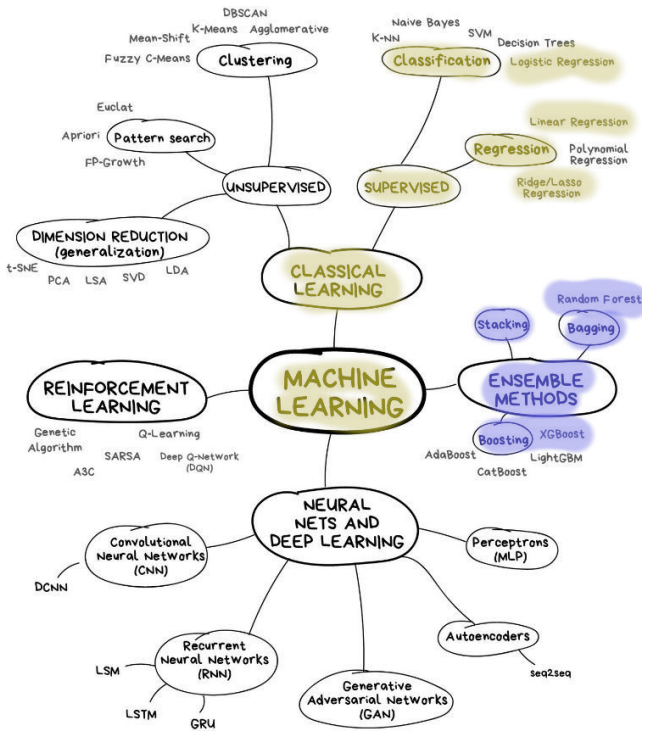
Learning Objectives

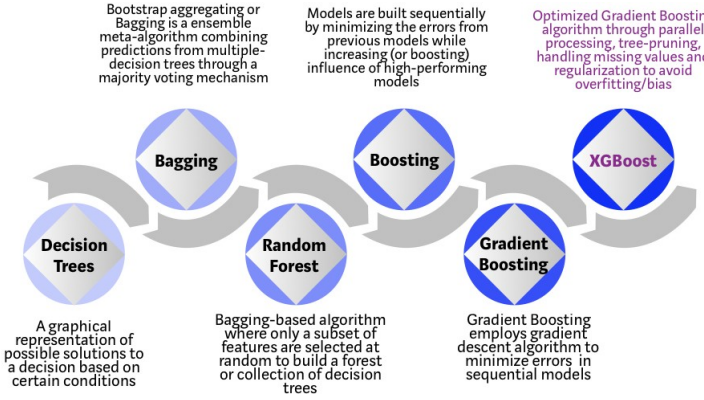
1. **Implement and evaluate machine learning pipelines.**
   - **Evaluate (find problems in) existing machine learning pipelines.**
   - **Design a pipeline to solve a given ML problem.**
   - **Implement some standard pipelines in Python.**
2. Implement and evaluate causal inference designs.
3. Understand how (not) to use data science tools (ML and CI) to support expert decision-making.


MACHINE LEARNING
CLASSICAL LEARNING
UNSUPERVISED
Clustering
DBSCAN
K-Means
Agglomerative
Mean-Shift
Fuzzy C-Means
Pattern search
Euclat
Apriori
FP-Growth
DIMENSION REDUCTION (generalization)
t-SNE
PCA
LSA
SVD
LDA
SUPERVISED
Classification
Naive Bayes
K-NN
SVM
Decision Trees
Logistic Regression
Regression
Linear Regression
Polynomial Regression
Ridge/Lasso Regression
ENSEMBLE METHODS
Stacking
Bagging
Random Forest
Boosting
XGBoost
AdaBoost
LightGBM
CatBoost
REINFORCEMENT LEARNING
Genetic Algorithm
Q-Learning
SARSA
Deep Q-Network (DQN)
A3C
NEURAL NETS AND DEEP LEARNING
Convolutional Neural Networks (CNN)
DCNN
Perceptrons (MLP)
Recurrent Neural Networks (RNN)
LSM
LSTM
GRU
Generative Adversarial Networks (GAN)
Autoencoders
seq2seq

Objectives in an ML Project

1. **What is the policy problem or research question?**
2. Data:
   - obtain, clean, preprocess, and link.
   - Produce descriptive visuals and statistics on the text and metadata
3. Machine learning:
   - Select a model and train it.
   - Fine-tune hyperparameters for out-of-sample fit.
   - Interpret predictions using model explanation methods.

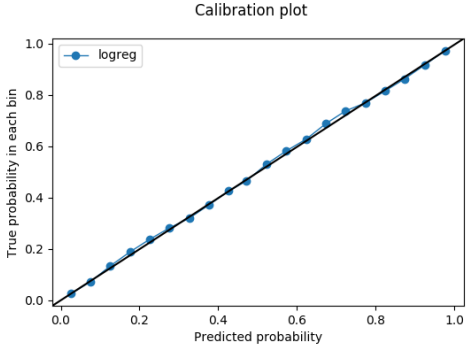

MACHINE LEARNING
CLASSICAL LEARNING
UNSUPERVISED
Clustering
DBSCAN
K-Means
Agglomerative
Mean-Shift
Fuzzy C-Means
Pattern search
Euclat
Apriori
FP-Growth
DIMENSION REDUCTION (generalization)
t-SNE
PCA
LSA
SVD
LDA
SUPERVISED
Classification
Naive Bayes
K-NN
SVM
Decision Trees
Logistic Regression
Regression
Linear Regression
Polynomial Regression
Ridge/Lasso Regression
ENSEMBLE METHODS
Stacking
Bagging
Random Forest
Boosting
XGBoost
AdaBoost
LightGBM
CatBoost
REINFORCEMENT LEARNING
Genetic Algorithm
Q-Learning
SARSA
Deep Q-Network (DQN)
ASC
NEURAL NETS AND DEEP LEARNING
Convolutional Neural Networks (CNN)
DCNN
Perceptrons (MLP)
Recurrent Neural Networks (RNN)
LSM
LSTM
GRU
Generative Adversarial Networks (GAN)
Autoencoders
seq2seq




```
from xgboost import XGBClassifier
model = XGBClassifier()

model.fit(X_train, y_train,
          early_stopping_rounds=10,
          eval_metric="logloss",
          eval_set=[(X_eval, y_eval)]
          )

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

Activity: Zoom Poll 6.1: True/False Quiz

Evaluating Classification Models: Calibration Curves



- Plotting the binned fraction in a category (Y axis) against the predicted probability in a category (X axis):
- Provides evidence of whether the classifer is replicating the conditional distribution of the outcome.

```
from seaborn import regplot
regplot(y_test, y_pred, x_bins=20)
```

Nested Sample Splits

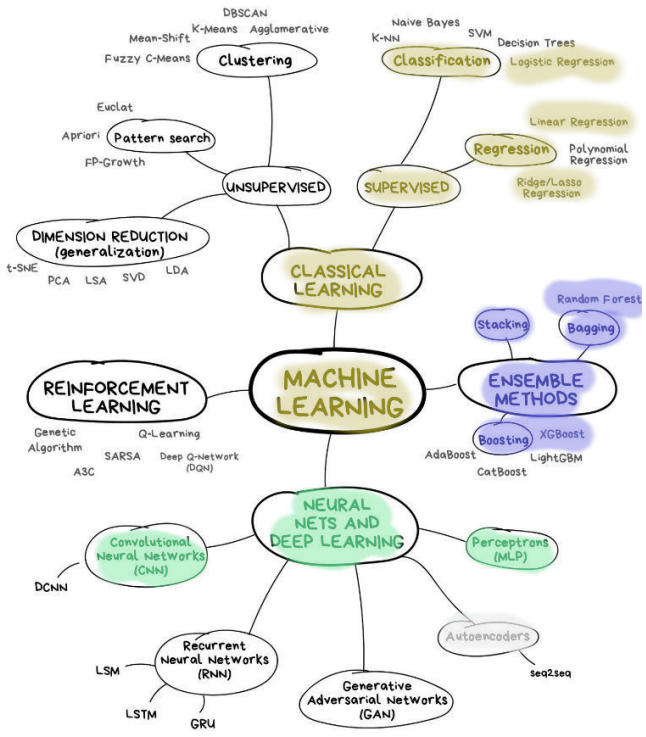
To do evaluations in the full data, use nested cross-validation:

- split data into K folds, e.g. 5.
- for each fold $k \in \{1, 2, ..., K\}$:
  - train and tune model in rest of data $\neg k$
  - evaluate metrics (e.g. MSE, balanced accuracy) in k.
- Report mean and s.d. of metrics across folds.

L2-regularized Logistic Regression in Keras

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

model.add(Dense(input_dim=num_features,
                units=2,  # output dimension
                activation='sigmoid',
                kernel_regularizer='l2',
                ))
```

```
 model.compile(optimizer='sgd', # stochastic gradient descent
               loss='binary_crossentropy',
               metrics=['accuracy'])

 model.fit(x_train, y_train,
           epochs=100,
           validation_data=(x_val, y_val))
```

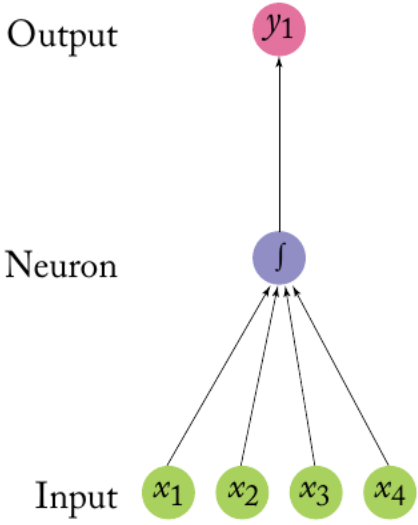

MACHINE LEARNING
CLASSICAL LEARNING
UNSUPERVISED
Clustering
DBSCAN
K-Means
Agglomerative
Mean-Shift
Fuzzy C-Means
Pattern search
Euclat
Apriori
FP-Growth
DIMENSION REDUCTION (generalization)
t-SNE
PCA
LSA
SVD
LDA
SUPERVISED
Classification
Naive Bayes
K-NN
SVM
Decision Trees
Logistic Regression
Regression
Linear Regression
Polynomial Regression
Ridge/Lasso Regression
ENSEMBLE METHODS
Stacking
Bagging
Random Forest
Boosting
XGBoost
AdaBoost
LightGBM
CatBoost
REINFORCEMENT LEARNING
Genetic Algorithm
Q-Learning
SARSA
Deep Q-Network (DQN)
A3C
NEURAL NETS AND DEEP LEARNING
Convolutional Neural Networks (CNN)
DCNN
Perceptrons (MLP)
Recurrent Neural Networks (RNN)
LSM
LSTM
GRU
Generative Adversarial Networks (GAN)
Autoencoders
seq2seq

- Neural networks $\leftrightarrow$ deep learning models
  - solve machine learning problems, just like logistic regression or gradient boosted machines
  - use tensorflow/keras or torch, rather than sklearn or xgboost.
- **why use neural nets?**
  - sometimes outperform standard ML techniques on standard problems
  - greatly outperform standard ML techniques on some problems, for example image recognition / text generation
- **why not use neural nets?**
  - usually worse than standard ML on standard problems, and harder to implement.
  - Computational constraints: Recent models like OpenAI's GPT-3 would take ETH Deep Learning Cluster 18 months to train.

“Neural Networks”, “Deep Learning”

- “**Neural**”:
  - NN’s do not work like the brain – such metaphors are misleading.
- “**Networks**”:
  - NNs are not “networks” as that is understood in mathematical network theory or social science.
- “**Deep**” Learning:
  - does not speak to profundity or effectiveness.
  - a banal origin, and a source of hype.

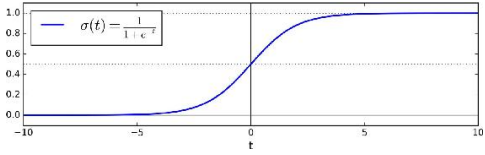
A "Neuron"



- applies dot product to vector of numerical inputs:
  - multiplies each input by a learned weight (parameter or coefficient)
  - sums these products
- applies a non-linear "activation function" to the sum
  - (e.g., the $\int$ shape indicates a sigmoid transformation)
- passes the output.

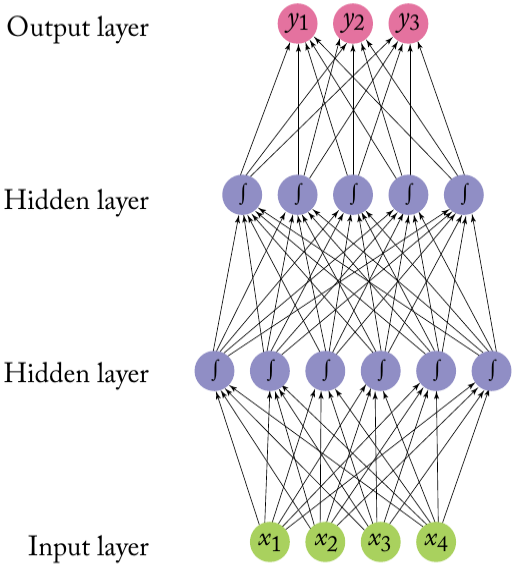
"Neuron" = Logistic Regression

$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$



- applies dot product to vector of numerical inputs:
  - multiplies each input by a learned weight (parameter or coefficient)
  - sums these products
- applies a non-linear "activation function" (sigmoid) to the sum
- passes the output.

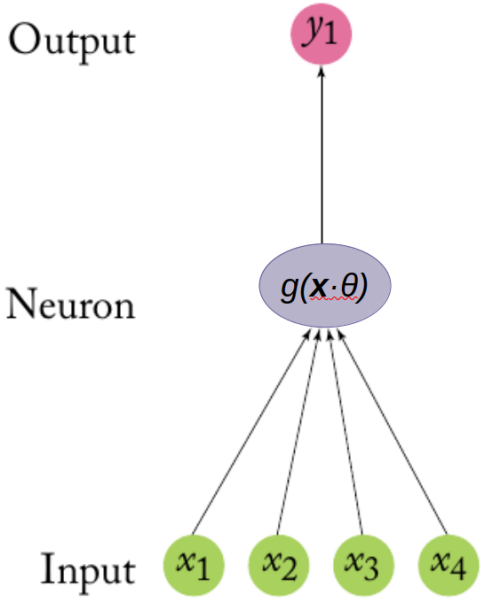
Feed-Forward Neural Network (FFN)



- A feed-forward network (also called a multi-layer perceptron or sequential model) stacks neurons horizontally and vertically.
- alternatively, think of it as a stacked ensemble of logistic regression models.
- this vertical stacking is the "deep" in "deep learning"!

- FFN's are composed of "Dense" layers – means that all neurons are connected.
- FFN with a single hidden layer, with sigmoid activation, can approximate any continuous function on a closed and bounded subset of $\mathbb{R}^n$, and any mapping from one finite discrete space to another finite discrete space (Hornik et al 1989, Cybenko 1989).
  - But NN would have to be exponentially large in some cases (Telgarsky 2016) .

Activation functions $g(\boldsymbol{x} \cdot \theta)$



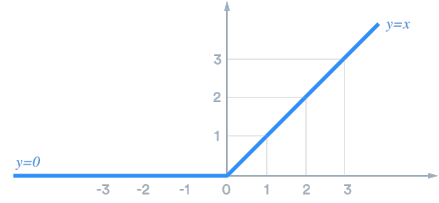
Previously we had

$$g(\boldsymbol{x} \cdot \theta) = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$

It turns out that sigmoid does not work well in hidden layers, mainly because gradient is flat except around zero.

ReLU (rectified linear unit) function:

$$g(\boldsymbol{x} \cdot \theta) = \text{ReLU}(\boldsymbol{x} \cdot \theta) = \max\{0, \boldsymbol{x} \cdot \theta\}$$

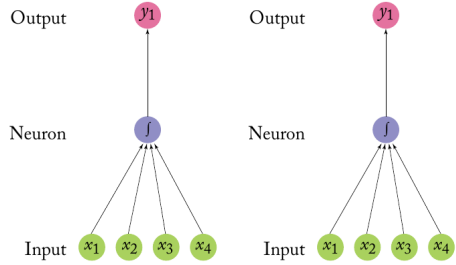


L2-regularized Logistic Regression in Keras

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

model.add(Dense(input_dim=num_features,
                units=2,  # output dimension
                activation='sigmoid',
                kernel_regularizer='l2',
                ))
```

```
print(model.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 2)                 10
=================================================================
Total params: 10
Trainable params: 10
Non-trainable params: 0
```



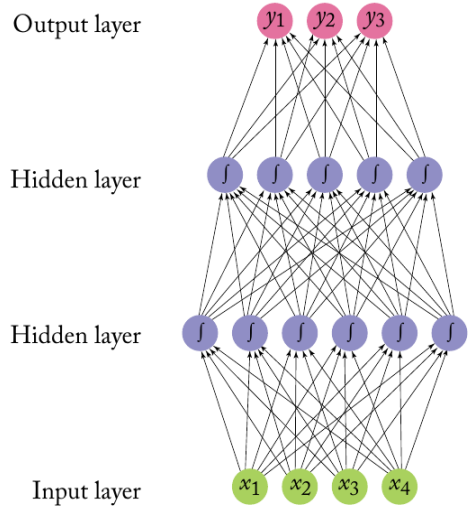
In this example, keras learns 10 parameters:

- coefficients on four predictors, plus a constant
- for each of two outcome classes

FFN in Keras

```
model = Sequential(name='FFN1')

# first hidden layer
model.add(Dense(6,  # neurons in layer
                input_dim=4,
                activation='relu'))

# second hidden layer
model.add(Dense(5,  # neurons in layer
                activation='relu'))

# output layer
model.add(Dense(3,  # output classes
                activation='softmax'))

print(model.summary())
```

```
Model: "FFN1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 6)                 30
_________________________________________________________________
dense_5 (Dense)              (None, 5)                 35
_________________________________________________________________
dense_6 (Dense)              (None, 3)                 18
=================================================================
Total params: 83
Trainable params: 83
Non-trainable params: 0
```



Coding Activity: Build an FFN

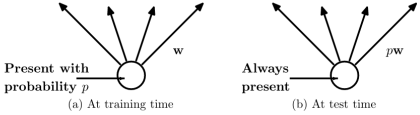
Early stopping

```
from tensorflow.keras.callbacks import EarlyStopping
earlystop = EarlyStopping(monitor='loss', patience=3)
model.fit(x_train, y_train,
          epochs=100,
          validation_data=(x_val, y_val)),
          callbacks=[earlystop])
```

As done with xgboost, a standard regularization approach for NNs is **early stopping:**

- Split data into three sets: training, validation, and test.
- stop training when validation-set loss stops improving
- evaluate model in test set.

Dropout



Source: Srivastava et al, JMLR 2014

- add after dense layers:

```
from tensorflow.keras.layers import Dropout
model.add(Dropout(.5))
```

An elegant regularization technique:

- at every training step, every neuron has some probability (typically $p = 0.5$) of being temporarily dropped out, so that it will be ignored at this step.
- at test time, neurons dont get dropped anymore but coefficients are down-weighted by p.

Why Dropout Works

- Approximates an ensemble of N models (where N is the number of neurons).
- Neurons cannot co-adapt with neighboring neurons and must be independently useful.
- Layers cannot rely excessively on just a few input neurons; they have to pay attention to all input neurons.
  - Makes the model less sensitive to slight changes in the inputs.

How to choose among so many options?

- the # of layers, # of neurons, regularization, dropout, etc are all tunable hyperparameters.
  - can pick these with cross-validation as we did previously.
- neural nets have many many dimensions for tuning.
  - this is a serious downside of neural nets, compared to the standard scikit-learn models.
- see the Geron book for advice on this point.
  - in general, make a big model (too many layers, too many neurons) and regularize with dropout/early stopping.

Overview (Sirignano, Sadhwani, & Giesecke 2018)

- Analyze mortgage risk using data from over 120 million loans for U.S. borrowers, 1995-2014
- Estimate deep learning model to predict loan status changes:
  - current; late; foreclosure
- Predictors:
  - loan variables at origination
  - loan performance variables over time
  - local economic variables

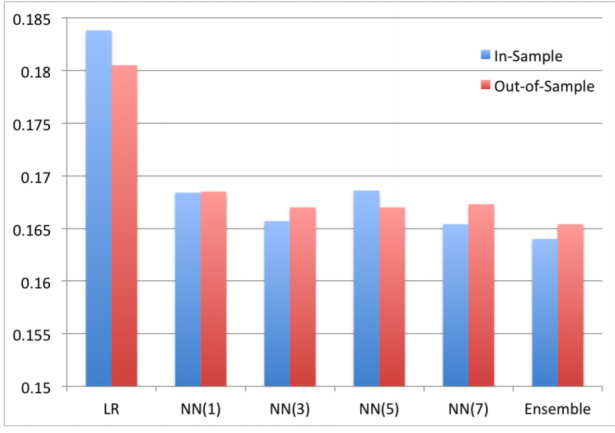
Monthly Transition Matrix (Outcome)

	Current	30	60	90+	Foreclosure
Current	97	1.4	0	0	.001
30 days	34.6	44.6	19	0	.004
60 days	12	16.8	34.5	34	1.6
90+ days	4.1	1.4	2.6	80.2	10
Foreclosure	1.9	.3	.1	6.8	87

Modeling

- Dataset is 350 billion loan-month transitions.
  - 294 predictors.
- Feed-forward network:
  - cross-validation picks 5 layers, ~200 neurons each, ReLU activation.
  - compare to logistic regression baseline

In- and out-of-sample errors vs. network depth

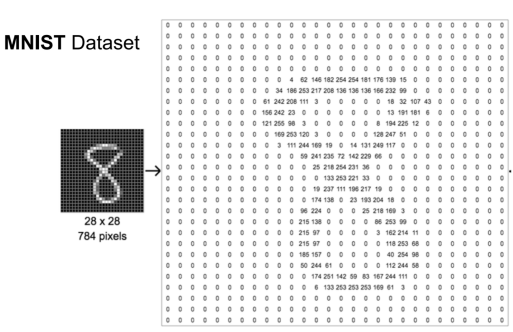


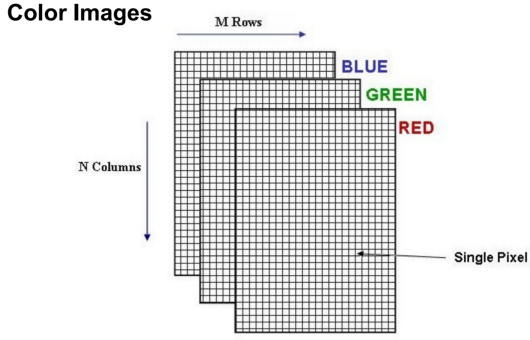
Global variable ranking by "leave-one-out"

Variable	**Test Loss**
State unemployment rate	1.160
Current outstanding balance	.303
Original interest rate	.233
FICO score	.204
Number of times 60dd in last 12 months	.179
Number of times current in last 12 months	.175
Original loan balance	.175
Total days delinquent ≥ 160	.171
Lien type = first lien	.171
Original interest rate - national mortgage rate	.170
LTV ratio	.169
Time since origination	.168
Debt-to-income ratio	.168
$\vdots$	$\vdots$

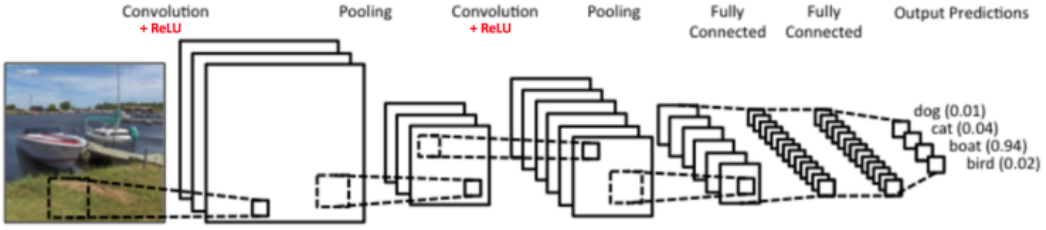
Computer Vision / Image Classification

- Most computer vision tasks can be understood as image classification:
  - taking an input image and output a probability distribution over classes that best describe the image.



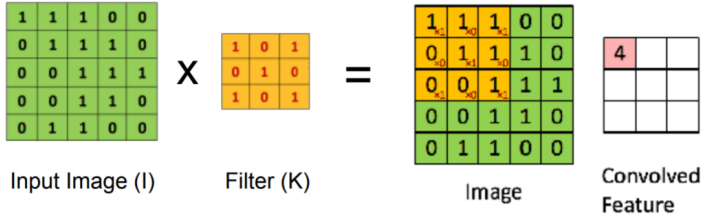


Convolutional neural nets



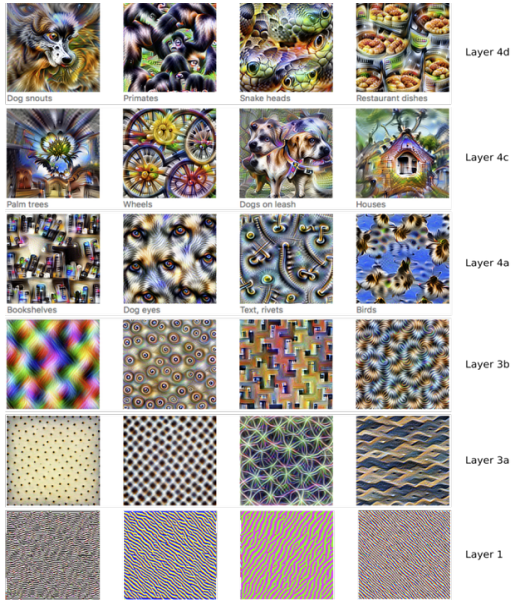
- CNNS are a special category of deep neural nets that have been especially effective at image classification.
  - they work by running filters over images to detect patterns which are predictive for a learning task.

Convolution Filters



- The 3×3 matrix K is a 'filter' (or 'kernel' or 'feature detector').
- A convolution slides the filter over the image, computing the dot product, producing the "feature map" (or "Convolved Feature" or "Activation Map").

How CNNs see the world



How CNNs see the world

- Convolutional neural nets learn these filters on their own during the training process.
  - we pick the number of filters, their size, etc.

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten

model = Sequential()
model.add(Conv2D(input_shape=(32, 32, 3), # 32x32 pixel, 3 color channels
                 filters=32, # no of filters
                 kernel_size=(3, 3), # filter size
                 activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Flatten()) # flattens matrix to vector
model.add(Dense(64, activation='relu'))
model.add(Dense(10)) # ten output classes
```

- In this course and most applications, I recommend transfer learning using a pre-trained model.

Transfer Learning with EfficientNet

- Can set up a minimal transfer learning model in just a few lines:

```
from tensorflow.keras.applications import EfficientNetB0
model = EfficientNetB0(input_tensor=X_preprocessed,
                       weights='imagenet',
                       include_top=False # for transfer learning
                       )
# add last layer
model.add(Dense(NUM_CLASSES, activation="softmax")
```

  - converts images to a vector that would be informative for image classification
  - last line adds a dense layer on top of this vector for the new task.
- For details see `http://bit.ly/BRJ-image-transfer`
  - e.g. data loading, image pre-processing, model training.

Overview (Ash and Borer 2020)

Objective:

- Dataset of video frames, with associated partisan labels (Democrat or Republican) from Wisconsin Ads Archive.
- Apply convolutional neural network classifer to predict partisanship based on image.
- Use trained model to analyze visual slant in videos from cable news networks.

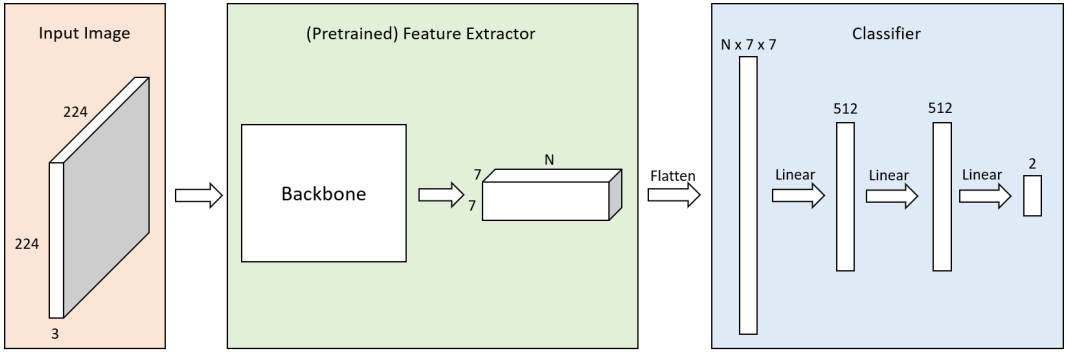
Labeled Video Frame Images

Democratic Party Candidates

Republican Party Candidates

- Dataset contains 17'619 video frames. Each frame is a full-color 319x240-pixel PNG image file.
  - 9'316 democratic (53%), 8'303 republican (47%)

Network Architecture



- Convolutional neural networks pretrained on large datasets work very well as feature extractors and also generalize well to different tasks.
- We experimented with different variants of VGG and ResNet, VGG-19 (20M parameters) did best.
  - for classifer, each linear layer followed by ReLU activation and dropout.

Image Processing and Perturbation

- Images resized to 224x224 pixels by scaling and center crop (padding resulted in worse performance).
- Brightness/saturation/etc normalized using statistics from ImageNet.
- We augment dataset by perturbing images slightly.
  - random rotations +-20 degrees
  - random translations +-10% of image size
  - random scaling +-30% of image size.
  - minor color variation to the pixel values, including brightness, contrast, saturation and hue.
- Helps the model learn relevant content, rather than specific visual style of corpus.

Sample Split

	Training (80%)	Validation (10%)	Test (10%)	Total
Democrat (53%)	7'452	930	934	9'316
Republican (47%)	6'641	829	833	8'303
Total	14'093	1'759	1'767	17'762

Training

- Model trained with early stopping (about 25 epochs).
- Observe accuracy and loss on training and validation sets, select model with lowest validation loss.
- Trained on Nvidia Titan Xp GPU, required about 150sec per epoch and 1 hour for the entire training.

Model Evaluation Metrics (Frame-Level Sampling)

	Accuracy	Precision	Recall	F1
Training	96.88%	95.31%	98.98%	97.1%
Validation	76.69%	74.71%	84.51%	79.3%
Test	**76.29%**	**74.04%**	**84.90%**	**79.1%**

Confusion matrix for test set

	Predicted Democratic	Predicted Republican
Actual Democratic	**44.88% (793)**	7.98% (141)
Actual Republican	15.73% (278)	**31.41% (555)**

Model Evaluation Metrics (Candidate-Level Sampling)

	Accuracy	Precision	Recall	F1
Training	83.75%	78.57%	95.26%	86.1%
Validation	58.15%	57.90%	76.24%	65.8%
Test	**58.18%**	**57.77%**	**76.44%**	**65.8%**

Confusion matrix for test set

	Predicted Democratic	Predicted Republican
Actual Democratic	40.24% (662)	12.40% (204)
Actual Republican	29.42% (484)	17.93% (295)

Example Images by Predicted Probability

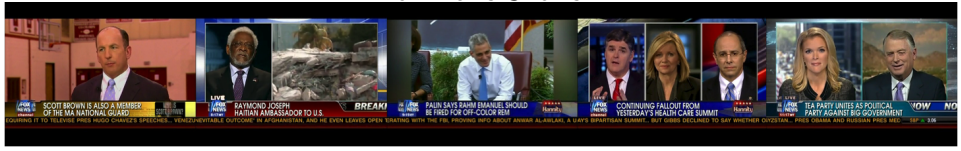
Images Most Likely to be Democrat

Images Most Likely to be Republican

Cable News Show Video

CNN Shows

Fox News Shows



MSNBC Shows

- 100 episodes from CNN, MSNBC, and FNC (Fox News Channel), 2009-2010. Sampled 180 seconds of video (starting at 16 minute mark). Extracted 10 images (once every 18 seconds).

Party Predictions for each TV News Network

- We applied the trained model to the processed images from TV Archive:

	Democrat	**Republican**
MSNBC	**76.12% (829)**	23.88% (260)
CNN	**68.96% (862)**	31.04% (388)
Fox News	33.24% (354)	**66.76% (711)**

Paper on Trustworthiness in Artworks

- Read this tweet thread (and click to paper for reference): `https://twitter.com/baumard_nicolas/status/1308715606196342784`
- Fork this google doc for your group: `http://bit.ly/BRJ-W6-A3-doc`
  - follow instructions there.