GAME OF CLASSIFICATION

Deep Learning on Game of Thrones Screenshots

Machine Learning II
Group 5
Michael Siebel

PURPOSE

- Game of Thrones was HBO's most popular TV series
  - It made HBO substantial profits
  - But, it was also the most pirated show in history
- My goal:
  - To create a copyright detection algorithm
  - Classify screenshots of Game of Thrones against a variety of other TV shows from another genre

A common issue television networks face is improper distribution of their intellectual property

CHALLENGE

- Game of Thrones is known for having a very high number of characters & sets
  - The model should not try to classify faces
  - Instead, general aesthetic
- Attributes of importance:
  - High production values
  - Color treated
  - Costume drama
  - Outdoor/rustic castles sets

COMPARISON GROUP

- All other shows are sitcoms
  - 13 sitcoms, *1989-Today*
  - Again, many characters & sets
  - Has sitcom aesthetic
- Attributes
  - Lower production values
  - Colorful/not color treated
  - Modern clothes/locations
  - Indoor sets

DATA

- Webscrapped all photos from the Internet Movie Database (IMDB.com)
  - All are screenshots (no BTS or promos)
  - Images were thumbnails
  - Only 100x100 pixels
- Pre-cropped photos
  - Focus more on faces
  - Backgrounds can be hidden
- Somewhat imbalanced data

Images

Total: 13,535

Game of Thrones: 3,883

Sitcom: 9,653

Game of Thrones comprises ~30% total data

MODELING

- Modeling with a Convolutional Neural Network (CNN)
  - Key attributes relate to RGB values
  - Image filters capture general aesthetic better than Multilayer Perceptron (MLP)
- Solving binary classification problem
  - Splitting data into training (70%), testing (15%), and validation (15%)
  - Training on a Binary Cross Entropy loss (BCELoss) function
  - Evaluating testing data on several models based on F1 scores
  - Evaluating final validation models based on F1 scores
- Coding is all done in Pytorch

DATA AUGMENTATION

- Oversampled Game of Thrones
  - Only on the training set
  - Doubled images
  - Shuffled into main training
- More balanced
  - Went from 29% to 44% of images

Labels	0 Sitcom	1 Game of Thrones
Pre-Oversampling	6763	2711
Post-Oversampling	6763	5422

```python
# Oversample GoT
x_ovsp = []
y_ovsp = []
for im, lab in zip(x_train, y_train):
    if lab == 1:
        x_ovsp.append(im)
        y_ovsp.append(lab)

y_ovsp = np.array(y_ovsp)

def shuffle_train(a, b):
    assert len(a) == len(b)
    shuffled_a = np.empty(a.shape, dtype=a.dtype)
    shuffled_b = np.empty(b.shape, dtype=b.dtype)
    permutation = np.random.permutation(len(a))
    for old_index, new_index in enumerate(permutation):
        shuffled_a[new_index] = a[old_index]
        shuffled_b[new_index] = b[old_index]
    return shuffled_a, shuffled_b

x_train = np.concatenate((x_train, x_ovsp), axis=0)
y_train = np.concatenate((y_train, y_ovsp), axis=0)
x_train, y_train = shuffle_train(x_train, y_train)
```

DATA AUGMENTATION

- Resized images to half their size
- On training:
  - Used proportional color jitter
  - Random flips and rotation
  - Normalized
- On testing:
  - Center crop
  - Normalized
  - Avoided randomized transformations

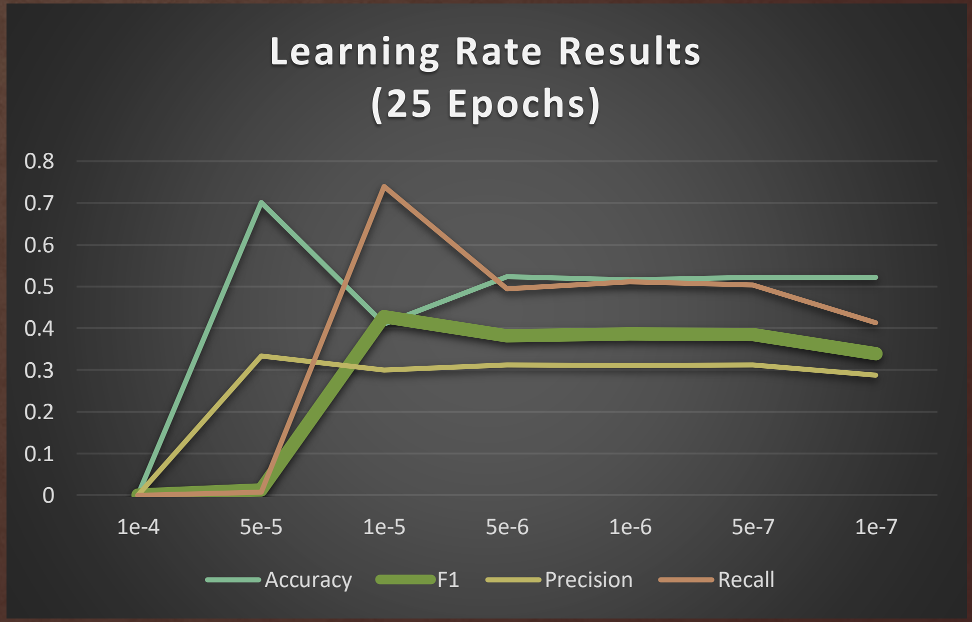
```python
# Data augmentation on photos
resize = 50

tf_train = transforms.Compose([
        transforms.RandomResizedCrop(resize),
        transforms.ColorJitter(.3, .3, .3),
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(20),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

tf_test = transforms.Compose([
        transforms.CenterCrop(resize),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

PARAMETERS AND NETWORK ARCHITECTURE

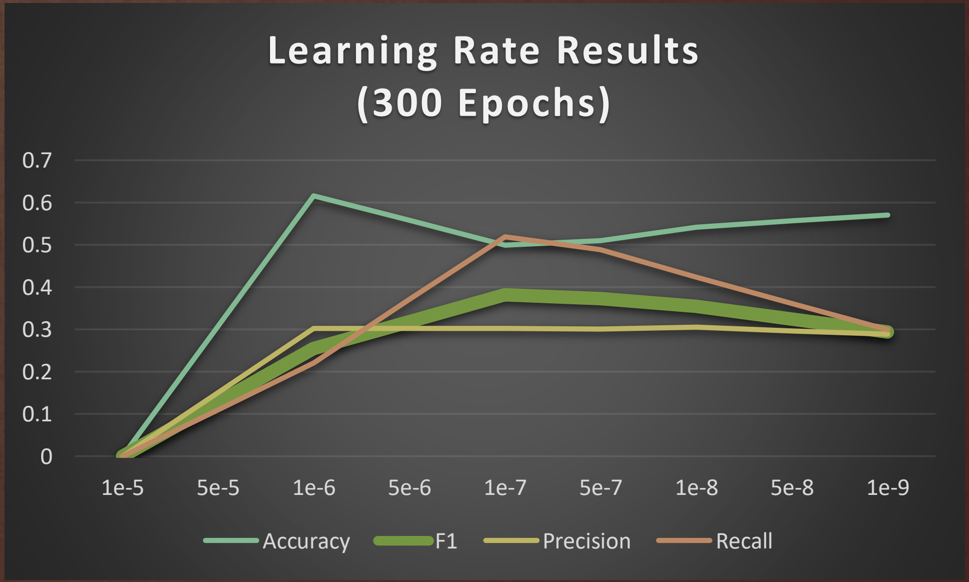
- Began with 5 convolutional layers
  - Batch normalization on each layer
  - 0.2 dropouts after layer 3, layer 5, and fully connected layer
- Oversampled (doubled) Game of Thrones
- Used only 25 epochs
- Evaluated best learning rate based on F1 score on test dataset



PARAMETERS AND NETWORK ARCHITECTURE

- Updated epochs to 300
  - Best learning rate became smaller
  - F1 scores did not necessarily improve
- Over predicting Game of Thrones
  - Oversampling is working too well

Confusion Matrix	Predicted Sitcom	Predicted Game of Thrones
Actual Sitcom	701	725
Actual Game of Thrones	290	314



FINAL MODEL ARCHITECTURE

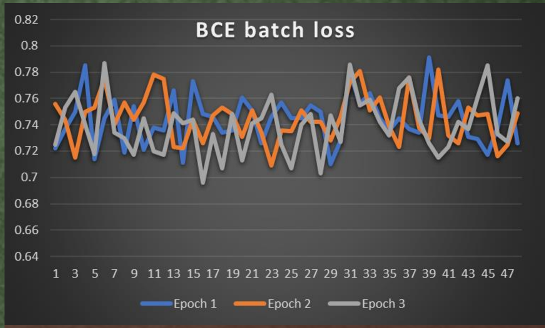
- Updated epochs to 500
- Updated learning rate to 1e-8
  - After attempting a learning rate of 1e-7
  - Received similar F1-score but more true positives with the lower rate
- Increased dropouts to 5x

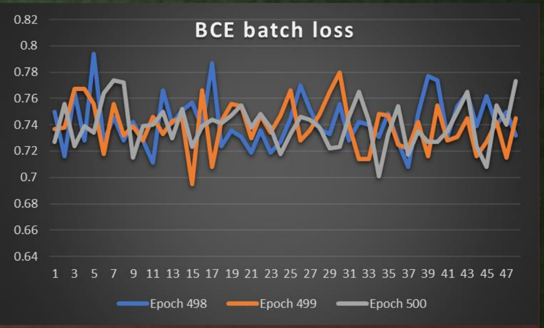
	Input	Output	Activation	Batch Normalization	Pooling	Dropout
Convolutional Layer 1	3	16	ReLU	2D	Max Pooling	
Convolutional Layer 2	16	32	ReLU	2D	Max Pooling	0.2
Convolutional Layer 3	32	64	ReLU	2D	Max Pooling	0.2
Convolutional Layer 4	64	128	ReLU	2D	Max Pooling	0.2
Convolutional Layer5	128	256	ReLU	2D	Max Pooling	0.2
Fully Connected Layer	256	128	ReLU	1D		0.2
Output layer	128	1	Sigmoid			

MAIN MODEL PERFORMANCE

- Ran on validation dataset
    - Achieved F1-score of 0.39
    - Achieved accuracy of 0.44
- Greatly over predicted
- Loss function did not improve over epochs

Confusion Matrix	Predicted Sitcom	Predicted Game of Thrones
Actual Sitcom	533	930
Actual Game of Thrones	202	366





ALTERNATIVE MODELS

- Experimented on the main model by
  - Removing oversampling
  - Reducing to 3 convolutional layers
  - Running a MLP version
- Surprisingly, the MLP version had similar classification scores as the main model
  - Achieved same F1-Score
  - Received slightly less true positives (342)

No Oversampling

F1-Score 0.29

Accuracy 0.60

Three Layers

F1-Score 0.35

Accuracy 0.53

MLP Version

F1-Score 0.39

Accuracy 0.47

CONCLUSION

- Learning rate was the most tuned hyper-parameter

  Did not do enough

- Epochs helped very little

  Allowed lower learning rates

  Did not lower the loss metrics much

- Oversampling only slightly helped

  Improved precision, lowered recall

- CNN surprisingly was not much better than MLP

  Image size was manageable by MLP

  Full sized versions would not be

- Layers make a difference

  Three layers performed much worse

  Small images, though, can only have so many pooling layers

NEXT STEPS

- Image quality was likely too weak for the complex analysis of which CNN’s are capable

  Would need to get full-size images for better analysis

- The loss function may be at issue

  Need to better investigate why the loss metrics did not diminish greatly

- Ensemble algorithms are likely necessary
- …Or simply try classifying the Simpsons