Towards Building An Effective Multi-Label Image Classification Model

Jadie | Lopez | Ponce

STINTSY S14

Outline

- The Task & The Dataset
- Data Preprocessing
- Simple EDA
- Model Training
- Model Selection & Hyperparameter Tuning
- Data Augmentation
- Challenges w/ the Data
- Insights & Conclusion

The Task & The Dataset

The Task

- **Multi-Label Image Classification**
  - *to associate multiple **labels** with a given image*
- Classify an image according to:

if it contains...

tom

jerry

both

neither

images from Kaggle (https://www.kaggle.com/datasets/balabaskar/tom-and-jerry-image-classification)

The Dataset

Tom and Jerry Image Classification

- Dataset containing images from various episodes of the famous cartoon show, Tom and Jerry.
- Currently contains **5,478** images (instances)

Source: Kaggle
https://www.kaggle.com/datasets/balabaskar/tom-and-jerry-image-classification

The Dataset

Tom and Jerry Image Classification

- A ground truth file (**ground_truth.csv**) is also provided containing the labels that correspond to each image file for supervised training.

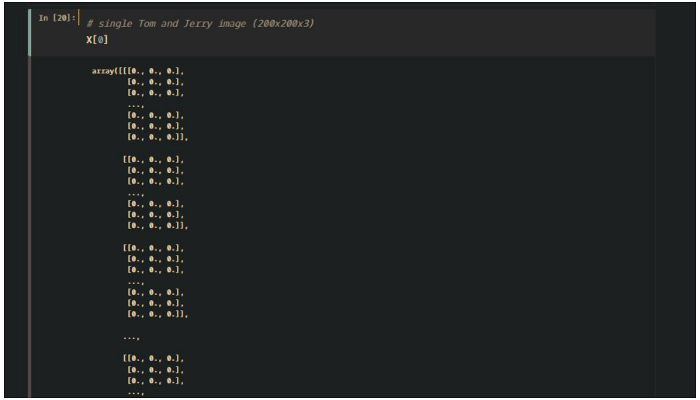
```
In [120]: # read ground_truth csv file and store it as a dataframe
          df_gt = pd.read_csv(ground_truth)

          # display the first 5 rows only
          df_gt.head()
```

	filename	tom	jerry
0	frame0.jpg	0	0
1	frame1.jpg	0	0
2	frame2.jpg	0	0
3	frame3.jpg	0	0
4	frame4.jpg	0	0

Data Preprocessing

Preprocessing

- All images are in JPEG format (1280x720).
  - Resized and converted into 3D NumPy **ndarrays** (224x224x3).
    - Faster training.
    - Defined fixed input size for the models that we are going to build and train.
  - Data type for each pixel value is **float32**.
    - Values are normalized, scaling them down to 0-1 from 0-255.
    - Faster convergence.



Preprocessing

- With the ground truth file read and converted into a dataframe, this is used to create the ground truth labels numpy (np) array which is “y”.
- Two np arrays were created: **X and y**.

```
In [214]: y

array([[0, 1],
       [0, 1],
       [0, 1],
       ...,
       [1, 1],
       [1, 1],
       [1, 1]])
```

```
In [117]: def create_img_dataset(img_dir, ground_truth, size=[224, 224]):
              # np ndarray containing the tom and jerry images represented 3D numpy ndarrays
              X = []
              # np ndarray containing the ground truth (y) labels for each image
              y = []

              # get ground truth labels and store them into a dataframe
              labels = pd.read_csv(ground_truth)

              # get and pre-process all images from each folder in the specified image
              for sub_dir in os.listdir(img_dir):
                  for img_file in os.listdir(os.path.join(img_dir, sub_dir)):
                      # get image path
                      img_path = os.path.join(img_dir, sub_dir, img_file)
                      # read image, it will be on the defaukt BGR format first
                      img = cv2.imread(img_path)
                      # convert to RGB format
                      img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                      # resize image according to set image dimensions
                      img = cv2.resize(img, (size[0], size[1]), interpolation=cv2.INTER_AREA)
                      # convert image to numpy array with float32 as data type
                      img = np.array(img).astype('float32')
                      # normalize the image array's values to 0-1
                      img /= 255
                      # store image in array
                      X.append(img)
                      # store image's ground truth class label, there must be a 'filename' column which
                      # corresponds to the name of the current image file being stored
                      y.append(labels[labels['filename'] == img_file].values[:, 1:].squeeze())

              # return the created image dataset as numpy ndarrays
              return np.array(X), np.array(y).astype(int)
```


```
# store image's ground truth class label, there must be a 'filename' column which
# corresponds to the name of the current image file being stored
y.append(labels[labels['filename'] == img_file].values[:, 1:].squeeze())
```

Simple EDA

Unique Values

- Unique values per row of **ground_truth.csv**

```
[0, 0] -> Neither
[0, 1] -> Jerry only
[1, 0] -> Tom only
[1, 1] -> Both
```

```python
In [6]: # get all unique rows (1D numpy arrays) and the number of images for each
        unique, num_of_imgs = np.unique(y, axis=0, return_counts=True)
```

Lets view the unique rows from the `y` numpy array.

```python
In [7]: unique
```

```
array([[0, 0],
       [0, 1],
       [1, 0],
       [1, 1]])
```

tom
[1,0]

Jerry
[0,1]

Both
[1,1]

Neither
[0,0]

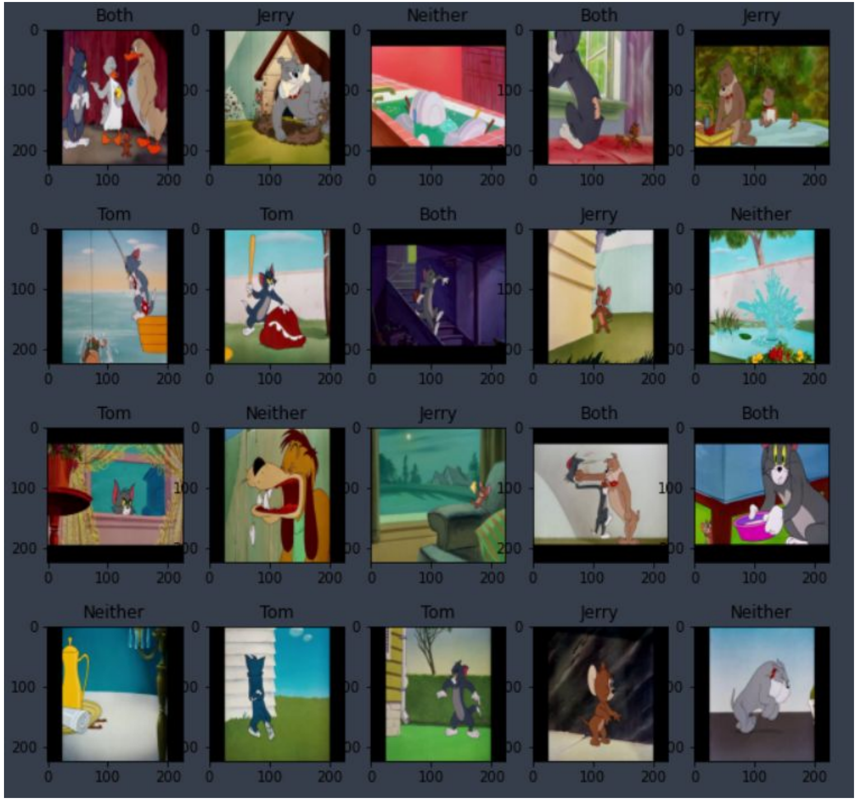
Distribution of Images

- A total of **5,478** images w/ the ff. distribution:

  1,528 -> Neither
  1,240 -> Jerry
  1,930 -> Tom
  780 -> Both

Some random samples

- Some things worth taking note of:
  - Scenes have **varying** contexts (i.e., different backgrounds, lighting, orientation)
  - In some, Tom and/or Jerry is even **physically distorted**.



Model Training

Splitting the Dataset

Set	Percentage (%)	Image Count
Training Set	80%	4,382
Validation Set	10%	548
Test Set	10%	548

Model of Choice

- **Model of Choice:** *Convolutional Neural Networks (CNNs)*
- **Specific Models Used:**
    - **AlexNet** (Krizhevsky et al, 2012)
        - Same architecture; trained from scratch
    - **ZFNet** (Zeiler & Fergus, 2014)
        - Same architecture; trained from scratch
    - **InceptionNetV3** (Szegedy et al., 2016)
        - Slightly modified architecture; with *transfer learning*

Model Accuracies

CNN	Train Accuracy (%)	Test Accuracy (%)
AlexNet	94.33%	88.96%
ZFNet	93.27%	85.95%
InceptionNetV3*	94.37%	84.58%

* with transfer learning

Model Accuracies

CNN	Train Accuracy (%)	Test Accuracy (%)
AlexNet	**94.33%**	**88.96%**
ZFNet	93.27%	85.95%
InceptionNetV3*	94.37%	84.58%

* with transfer learning

Model Selection and Hyperparameter Tuning

Hyperparameter Tuning

- **Keras Tuner** was used to conduct hyperparameter tuning for each CNN model.
- With Keras Tuner, the **random search** method was used.

For each model:

- Did not touch and tune the original architecture (layers) themselves.
- AlexNet -> **Learning rate** of the Adam optimizer
- ZFNet -> **Learning rate** of the Adam optimizer
- InceptionNetV3 -> **Learning rate** of the Adam optimizer and **dropout rate** of the last dropout layer.

Hyperparameter Tuning: AlexNet

```python
# hp here is the parameter passed by Keras Tuner for tuning the specified hyperparameters
def alex_net_tuner(hp):
    alexnet = alex_net()

    # tune the learning rate of the adam optimizer
    # we use hp.Choice() to define the learning rate values to be used for each training trial
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

    alexnet.compile(optimizer=Adam(learning_rate=hp_learning_rate),
                    loss='binary_crossentropy',
                    metrics=['binary_accuracy'])

    return alexnet
```

```python
tuner_alexnet = kt.RandomSearch(
    alex_net_tuner, objective='val_loss', directory='tuned_models', project_name='tuned_alexnet')
```

Hyperparameter Tuning: ZFNet

```
def zf_net_tuner(hp):
    zfnet = zf_net()

    # tune the learning rate of the adam optimizer
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
    zfnet.compile(optimizer=Adam(learning_rate=hp_learning_rate),
                  loss='binary_crossentropy',
                  metrics=['binary_accuracy'])

    return zfnet
```

```
tuner_zfnet = kt.RandomSearch(
    zf_net_tuner, objective='val_loss', directory='tuned_models', project_name='tuned_zfnet')
```

Hyperparameter Tuning: InceptionNetV3

```python
def inceptionv3_tuner(hp):
    pretrained = InceptionV3(weights="imagenet", include_top=False, input_shape=(224, 224, 3))

    for layer in pretrained.layers:
        layer.trainable = False

    model = pretrained.output
    model = MaxPool2D(pool_size=(5,5), strides=(2,2))(model)
    model = Flatten()(model)
    model = Dense(4096, activation="relu")(model)

    # tune the dropout rate
    hp_dropout = hp.Float('rate', min_value=0.1, max_value=0.5, step=0.1)
    model = Dropout(hp_dropout)(model)

    output = Dense(2, activation="sigmoid")(model)

    inceptionv3 = Model(inputs=pretrained.input, outputs=output)

    # tune the learning rate for the adam optimizer
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

    inceptionv3.compile(optimizer=Adam(learning_rate=hp_learning_rate),
                        loss='binary_crossentropy',
                        metrics=['binary_accuracy'])

    return inceptionv3
```

```python
tuner_inceptionv3 = kt.RandomSearch(
    inceptionv3_tuner, objective='val_loss', directory='tuned_models', max_trials=20,
    project_name='tuned_inceptionv3')
```

Summary of Results: Model Selection

Model	Number of Correct Predictions	Test Accuracy
AlexNet	487.5 / 548	88.9599%
ZFNet	471.0 / 548	85.9489%
InceptionNetV3	463.5 / 548	84.5802%

Tuned Model	Best Optimizer Learning Rate	Best Dropout Rate	Number of Correct Predictions	Test Accuracy
AlexNet	0.0001	(Not tuned)	510.0 / 548	93.0656%
ZFNet	0.001	(Not tuned)	461.0 / 548	84.1240%
InceptionNetV3	0.0001	0.5	481.0 / 548	87.7737%

Data Augmentation

Data Augmentation

- **ImageDataGenerator** of **Keras** was used.
- The **train** set's data was augmented by:
    - **Rotating**
    - **Shifting (Width and Height)**
    - **Zooming**
    - **Flipping (Horizontally and Vertically)**

```
train_datagen = ImageDataGenerator(rotation_range=5,  # rotation
                                   width_shift_range=0.01,  # horizontal shift
                                   height_shift_range=0.01, # vertical shift
                                   zoom_range=0.2,  # zoom
                                   horizontal_flip=True,  # horizontal flip
                                   vertical_flip=True,  # vertical flip
                                  )
```

Data Augmentation: Results

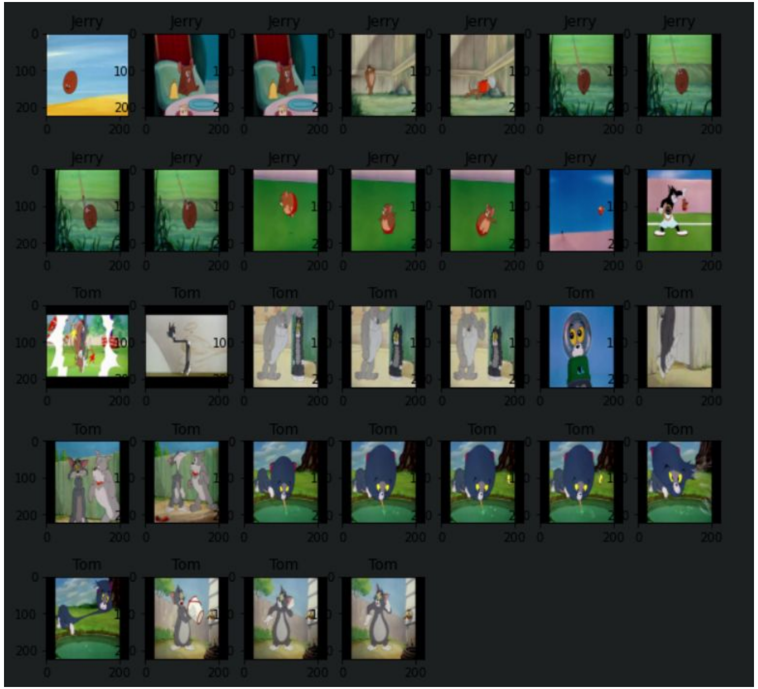
Model	Number of Correct Predictions	Accuracy
Initially Trained AlexNet	487.5 / 548	88.9598%
Tuned AlexNet	510.0 / 548	93.0656%
Tuned AlexNet w/ Data Augmentation	518.0 / 548	94.5255%

Challenges with the Data

Challenge Dataset

- There are images wherein Tom and Jerry are distorted in **shape, size, and color**.
- A **challenges csv file** was added which contains the list of **some distorted images** found in the dataset.

	folder	image_name
0	jerry	frame360.jpg
1	jerry	frame1794.jpg
2	jerry	frame1795.jpg
3	jerry	frame2231.jpg
4	jerry	frame2234.jpg



Challenge Dataset: Results

```
In [207]: scores = best_model.evaluate(X_ch, y_ch)
```

```
1/1 [==============================] - 1s 711ms/step - loss: 0.0064 - binary_accuracy: 1.0000
```

Let's get the best model's number of correct predictions on the challenge dataset.

```
In [208]: predictions = best_model.predict(X_ch)

          predictions
```

```
       [4.1455085e-08, 9.9930751e-01],
       [2.7008427e-09, 9.9999905e-01],
       [7.0454651e-09, 9.9999970e-01],
       [1.9435412e-03, 9.9195880e-01],
       [9.5749790e-05, 9.5904613e-01],
       [9.9968618e-01, 3.8940880e-02],
       [9.9789143e-01, 2.0948276e-02],
       [1.0000000e+00, 5.6303469e-07],
       [9.9999982e-01, 1.7406048e-05],
       [1.0000000e+00, 6.4829583e-06],
       [9.9968362e-01, 1.0855265e-03],
       [9.9958909e-01, 1.8719690e-04],
       [9.9999183e-01, 1.4778913e-04],
       [1.0000000e+00, 3.7413497e-07],
       [9.9990690e-01, 7.6467968e-03],
       [9.9999279e-01, 6.2163710e-03],
       [9.9993569e-01, 1.9419657e-02],
       [9.9993378e-01, 2.8760826e-02],
       [9.9773252e-01, 7.3010395e-03],
       [9.9999970e-01, 7.8685865e-05],
       [9.9985713e-01, 2.1912361e-04],
       [1.0000000e+00, 1.2117609e-04],
       [1.0000000e+00, 2.1450575e-04]], dtype=float32)
```

```
In [209]: correct = compute_correct(predictions, y_ch)

          print("Number of correct predictions: {} / {}".format(correct, len(y_ch)))
          print("Test accuracy: {}".format(scores[1]))
```

```
Number of correct predictions: 32.0 / 32
Test accuracy: 1.0
```

Insights & Conclusion

Some Insights & Conclusion

Insight #1: Hardware constraints. Training on machines not suited for deep neural networks is a tedious task. Training time takes hours, and sometimes, some deep learning models do not train at all.

Insight #2: Data Augmentation helps. This greatly helps in improving the performance of a CNN model given a small dataset.

Insights & Conclusion

Insight #3: More challenges on the data. Best model currently finds it difficult to correctly predict the labels on these kinds of images:

Some Insights & Conclusion

Insight #4: CNNs are powerful model architectures for image classification. Future goal to construct and train more CNN models that are also better in performance such as **VGG-19**, **ResNet**, **DenseNet**, **and Efficient.**

References:

Baskar, B. (n.d.). Tom and Jerry Image Classification. Kaggle. Retrieved June 23, 2022, from https://www.kaggle.com/datasets/balabaskar/tom-and-jerry-image-classification

Brownlee, J. (2019). A Gentle Introduction to Transfer Learning for Deep Learning. Retrieved June 22, 2022, from https://machinelearningmastery.com/transfer-learning-for-deep-learning/

Draelos, R. (2019). Multi-label vs Multi-class Classification: Sigmoid vs. Softmax. Retrieved June 21, 2022, from https://glassboxmedicine.com/2019/05/26/classification-sigmoid-vs-softmax/

Dwivedi, R. (2020). How Data Augmentation Impacts Performance of Image Classification, With Codes. Retrieved June 25, 2022 from https://analyticsindiamag.com/image-data-augmentation-impacts-performance-of-image-classification-with-codes/

Keras (n.d.). Keras Documentation: Keras API reference. Keras. Retrieved June 26, 2022, from https://keras.io/api/

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional Neural Networks. Communications of the ACM, 60(6), 84–90. https://doi.org/10.1145/3065386

Lang, N. (2021). Using Convolutional Neural Network for Image Classification. Retrieved June 26, 2022, from https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4

Migmar, T. (2021). Understanding the Amazon Rainforest with Multi-Label Classification + VGG-19, Inceptionv3, AlexNet & Transfer Learning. Retrieved June 24, 2022, from https://towardsdatascience.com/understanding-the-amazon-rainforest-with-multi-label-classification-vgg-19-inceptionv3-5084544fb655

Mustafeez, A. Z. (n.d.). What is early stopping? Retrieved June 23, 2022, from https://www.educative.io/answers/what-is-early-stopping

Rausch, D. (2021). EDA for Image Classification. Retrieved June 24, 2022, from https://medium.com/geekculture/eda-for-image-classification-dcada9f2567a

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826).

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham.

Towards Building An Effective Multi-Label Image Classification Model

Jadie | Lopez | Ponce
STINTSY S14