

Applying NLP Deep Learning Ideas to Image Classification

Gary Ren

SCPD Student at Stanford University

Applied Scientist at Microsoft

garyren@stanford.edu, gren@microsoft.com

Abstract

NLP deep learning models have had great success in recent years by using word embeddings, RNNs, and attention mechanism. This paper presents several models that leverage these ideas together with CNNs for the task of multi-label image classification. Some of these models produced state of the art results on the MS-COCO dataset, demonstrating the benefit of leveraging these NLP deep learning ideas, which was also confirmed by further analysis of the results.

1. Introduction

Image classification is the task of classifying images given a set of discrete labels, a core task in computer vision with tons of applications. Some of the challenges of image classification include viewpoint variation, illumination variation, deformation, occlusion, background clutter, and intraclass variation. There are several image classification datasets and in recent years, novel deep learning (specifically convolutional neural network) models have continuously achieved new state of the art results on these datasets. This paper presents attempts at further pushing the state of the art in image classification by borrowing ideas from NLP deep learning techniques.

1.1. Multi-label image classification

Traditionally, image classification has focused on assigning a single class to each image. However, in many real world images, there are multiple different objects in an image and it is important to be able to identify all the different objects. Therefore, this project focused on the task of multi-label image classification. Specifically, given an input image, the goal is to output a list of all the different classes that appear in that image.

1.2. Deep learning for NLP

Besides computer vision, NLP is another area where deep learning has led to great progress in recent years.

The first step in most NLP deep learning models is to convert the words into vector representations that can be passed as input to a neural network. These vector representations are commonly known as word embeddings, and they capture the semantic meaning of words [10]. For example, semantically similar words will be close together in the vector space. Depending on the NLP task/model, the word embeddings are either: 1) set to pretrained embeddings, 2) initialized to pretrained embeddings but made trainable so that they can be shifted as necessary, or 3) trained from scratch.

Similar to how convolutional neural networks (CNNs) is the most common fundamental architecture used for computer vision, recurrent neural networks (RNNs) is the most common architecture for NLP. RNN cells such as the Gated Recurrent Unit (GRU) [4] and Long Short Term Memory (LSTM) [16] are commonly used to help maintain information from previous time steps and improve gradient flow.

Another popular technique used in NLP is the attention mechanism, which basically allows RNN models to more directly look at multiple previous time steps instead of just the one previous time step. The attention is computed over multiple previous time steps via some scoring function, and can be used to augment the outputs from RNNs [18] or directly used at the output layer to "point" to a previous time step [2].

Word embeddings, RNNs, and attention are the three ideas that are leveraged for this project.

2. Related Work

The following subsections will go over the existing applications of word embeddings, RNNs, and attention in computer vision.

2.1. Word embeddings

For the tasks of image captioning, image question answering, and other computer vision tasks where the output is a sequence of words, word embeddings are often used for generating the output sequences. For image classification, there have been just a few applications of word embeddings. Wang et al. used word embeddings for multi-label image

classification [5]. They leveraged word embeddings and RNNs to capture the co-occurrence dependencies between labels (e.g. "sky" and "cloud" often appear together). Akata et al. used word embeddings for zero shot image classification (where some classes don't have any labeled samples in training set) [13]. They built upon the work of attributed based classification and used Word2Vec to help learn the relationships between classes, and also used the idea of embeddings by creating vector representations of the attributes.

2.2. RNNs

For terminology, let's consider deep learning models to have two main components: 1) an encoder that takes the input and encodes it as some hidden state, and 2) a decoder that takes the encoder output and makes predictions. Similar to word embeddings, RNNs have been used in the decoder for computer vision tasks where the output is a sequence of words. As mentioned earlier, Wang et al. used RNNs to generate multi-label image classifications. The model presented by Xu et al. [7] is an example of RNNs used for image captioning, and the model by Zhu et al. [12] is an example of RNNs used for image question answering.

However, I was not able to find examples of RNNs being directly used in the encoder of computer vision models. The closest existing work are image classification models that leverages the same idea of improving gradient flow. The most well known example is residual net (ResNet) [6]. ResNet adds shortcuts between CNN layers so that information from previous layers is directly passed and the later layers just have to learn the residual information. This also helps the gradient to flow back to lower layers, similar to in RNNs. Another example is the work by Srivasta et al., which specifically stated that they drew inspiration from LSTMs for the highway networks that they implemented [22]. Highway networks add a transform gate and carry gate to each layer. The transform gate determines how much of the final output of that layer is produced by transforming the input while the carry gate determines how much is produced by carrying the input.

2.3. Attention

Attention has been utilized in computer vision models for attending to specific areas in the input image. The image captioning model by Xu et al. [7] and the image question answering model by Zhu et al. [12] both obtained improvements by adding spatial attention. A similar idea to attention was used by GoogLeNet [3], which added auxiliary output layers at lower layers, in some sense adding attention to the lower layers, allowing the model to directly look at previous layers for classification.

3. Methods

3.1. Image-label joint embedding

I first implemented the model presented by Wang et al. [5] for multi-label image classification, which uses a combination of CNN and RNN. The CNN used is the VGG16 model [21] pretrained on ImageNet. The RNN is the decoder that outputs the predicted labels one at a time. A RNN is used because the hypothesis is that the labels for a given image have co-occurrence dependency, e.g. if a picture contains a keyboard, it likely also contains a mouse. LSTMs [16] are used and the input at each step is the word embedding of the previous predicted label. Note that during training, the input at each step is the word embedding of the previous correct label. This is so that the model can still learn the dependency between the label at t and $t - 1$ even if the prediction for $t - 1$ was incorrect. The output of the LSTM cell and the output of the CNN are both projected to the same dimension as the label embeddings and summed together to form an image-label joint embedding:

$$x_t = \tanh(h_t W_h + I W_I) \quad (1)$$

h_t is the output of the LSTM cell at time t and I is the output of the CNN. The projection $W_I I$ replaced the final fully connected layer of VGG16, and therefore needed to be re-trained. Note that this equation (and subsequent equations in this paper) is based on batch major inputs and outputs. The intuition behind creating a joint embedding is that by using this embedding for prediction, the model is predicting labels that are conditioned on both the previous labels and the image, allowing the model to capture both the label dependencies and the image-label relevance.

This joint embedding is compared with each label embedding by computing the distance:

$$s_t = x_t E^T \quad (2)$$

The predicted label probabilities is just the softmax of these distance scores. Note that vectors that are closer together will have larger dot products, and therefore larger distance scores and larger label probability. The loss function is cross entropy loss on the label probabilities. A high level view of the model architecture is shown in figure 1.

3.2. CNN as initialization

While the image-label joint embedding model considers both the previously predicted labels and the image when making each prediction, the representation of the image that it uses is the same at each time step. This is not ideal because different image features might be important when predicting different labels, so having different representations of the image at each time step might help with predicting the correct label at each time step. In other words, the image

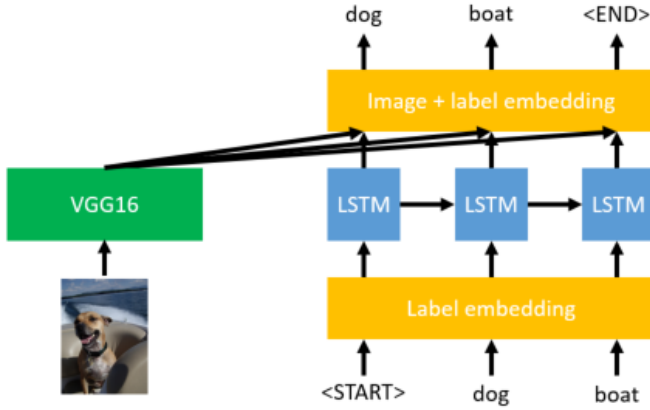


Figure 1. Image-label joint embedding

representation should also be conditioned on the previously predicted labels. This can be implemented by leveraging the ability of RNNs to maintain a hidden state that passes on information from previous time steps and adds new information as needed. By initializing the hidden state of the RNN with the image representation, the RNN can pass on the image representation and modify it as needed when predicting each label. A high level view of the model architecture is shown in fig 2. Note that this model is similar to many image captioning models, such as the model we implemented for assignment 3 and the model from Xu et al. [7].

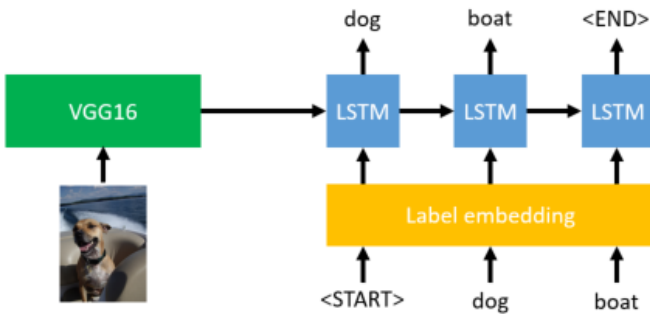


Figure 2. CNN as initialization

Again, the image representation is obtained from the pre-trained VGG16 model, with the final layer modified and re-trained to project to the proper dimensions for initializing the RNN. LSTMs cells were again used for the RNN. I implemented with three different ways of initializing the RNN: 1) have the CNN output a projection of hidden state size and use as the first hidden state with zero cell state 2) have the CNN output two different projections of hidden state size and use as the first hidden and cell states 3) have the CNN output a projection of embedding size and use as the input of an additional LSTM cell, whose outputted hidden and

cell states are used as the first hidden and cell states.

3.3. CNN sequence to sequence

GoogLeNet [3] obtained accuracy improvement by adding classification layers to the intermediate layers, proving that lower level image features can also be useful for classification. However, GoogLeNet assigned an arbitrary importance to the lower layers by discounting them by a set weight of 0.3.

In order to create a model that can learn the importance of each intermediate layer and have this importance be conditioned on the input image, the ability of RNNs to maintain a hidden state is again leveraged. In addition to the decoder RNN that outputs the predicted labels, a new encoder RNN is added that takes as inputs the intermediate layers of the CNN (still VGG16). This encoder RNN learns to maintain the information from the lower layers in such a way that optimizes the classification loss function, i.e. information from the lower layers will be passed via the hidden states and maintained as needed. This model is similar in structure to the sequence to sequence (seq2seq) model [23] that is commonly used in NLP for tasks where the input and output are both sequences. In this case, instead of an input sequence of text, the input sequence is the different levels of image features. Specifically, the features after each pooling layer and the final layer is projected into the embedding space and fed as inputs into the encoder RNN. A high level view of the model architecture is shown in figure 3.

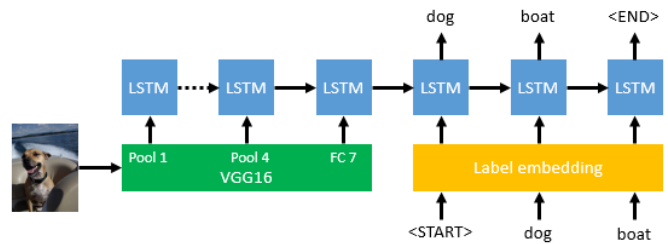


Figure 3. CNN sequence to sequence

3.4. Adding attention

Attention over the encoder states allows the decoder to have more direct access to the information from earlier time steps. In this case, the information from earlier time steps are the lower level image features. Therefore, I implemented attention on the CNN sequence to sequence model to give it more direct access to the lower level image features. Attention works by first computing an attention score for each of the encoder time steps. This score is conditioned on the current decoder output and the encoder output for each time step. Several different scoring functions have been used for existing attention models [18]. I chose the scoring function first presented by Bahdanau et al.[1],

which is one of the more commonly used scoring functions, and is referenced in several papers such as [18], [8], and [9]. This scoring function computes the attention score a_i^t for decoding step t and encoding step i using the following equations:

$$s_i^t = \tanh(h_i W_1 + d_t W_2 + b) v^T \quad (3)$$

$$a_i^t = \text{softmax}(s_i^t) \quad (4)$$

h_i is the encoder output at step i and d_t is the decoder output at step t . The attention scores are then used to compute a context vector that is the weighted sum of all the encoder outputs:

$$c_t = \sum_i a_i^t h_i \quad (5)$$

This context vector is then concatenated with the decoder output to create an attentional output/hidden state (recall that the output for a LSTM cell is its hidden state, hence the interchangeable terminology):

$$\tilde{d}_t = \tanh([c_t; d_t] W_c + b_c) \quad (6)$$

This attentional output state is what is passed to the final output projection layer to get the label probability distributions. A high level view of the model architecture can be seen in figure 4.

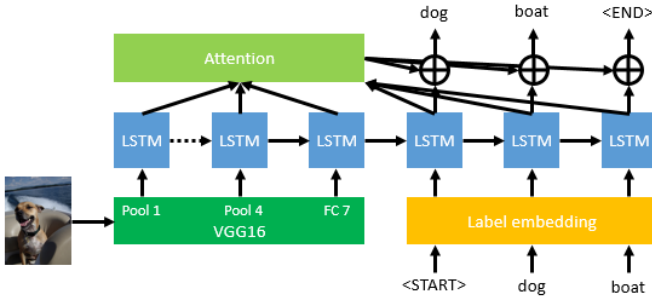


Figure 4. CNN sequence to sequence with attention

To make each decoding step aware of the attention from the previous decoding step, the input feeding approach [18] was also implemented. The previous attentional output state is concatenated with the next input to create the input that is passed to the LSTM:

$$\tilde{x}_t = \tanh([\tilde{d}_{t-1}; x_t] W_x + b_x) \quad (7)$$

In total, the new learnable parameters added by the attention mechanism are W_1 , W_2 , b , v , W_c , b_c , W_x , and b_x .

I also implemented the attention mechanism on the CNN as initialization model and the image-label joint embedding model by directly considering the outputs from the intermediate CNN layers as the encoder output states (h_i in the

above equations). For the CNN as initialization model, the rest of the attention mechanism is identical to what has been described. For the image-label joint embedding model, the context vector is computed in the same way, but instead of then computing an attentional output state, the context vector is used as the image representation I in equation (1). As a quick side note, the code referenced when implementing these models are [19], [24], and [25].

4. Dataset

4.1. MS-COCO

The dataset used for this project is the Microsoft Common Objects in Context (MS-COCO) dataset [11]. This dataset contains labels for image classification, segmentation, and captioning. For image classification, there are >123k images and 80 object categories, covering a wide range from dogs to airplanes to pizza. MS-COCO was chosen for this project because it has 3 different objects per image on average, making it well suited for the task of multi-label image classification. Table 1 shows a few sample images and table 2 shows the top five most and least frequent classes in the full MS-COCO train set. Existing work on multi-label image classification also used MS-COCO [5], making it possible to compare my results with existing results. In order to speed up training and evaluation time, especially since there were several different models implemented for this project, a subset of MS-COCO was used. 20k images were sampled for the train set, 5k for validation set, and 5k for test set.

4.2. Data preprocessing

For image preprocessing, I used the code provided by a TA of the class, Olivier Moindrot [19]. The code implements the standard preprocessing for VGG on ImageNet [21]. This same preprocessing was used because my models used VGG as their CNN and also because MS-COCO images have similar sizes to ImageNet images. The images are first resized so that their smaller side is 256 pixels. Each channel is normalized by subtracting its mean. During training, a random 224x224 crop is taken and the image is flipped horizontally with 50% probability. During evaluation, the center 224x224 crop is taken and the image is never flipped.

For the labels, they were preprocessed so that only the unique labels for each image remain. In MS-COCO, if there are multiple instances of the same class in an image, then there will be multiple labels for that class, which is needed for segmenting the different instances; but since this project is on classification, the duplicate labels were removed. Another preprocessing step required for the labels is to set their order for each image. Since the models that I implemented use RNNs to output labels one at a time, the order of the

Image	Labels
	dog, boat
	person, car, stoplight, backpack, handbag, tie, suitcase, cellphone
	zebra, elephant, giraffe
	pizza, fork, cup, table

Table 1. Samples from MS-COCO

Most Frequent	Count	Least Frequent	Count
person	45174	hair drier	128
chair	8950	toaster	151
car	8606	parking meter	481
dining table	8378	bear	668
cup	6518	scissors	673

Table 2. Most and least frequent classes from MS-COCO

labels used during training is an important consideration. I decided to order the labels by their frequency in the train set, from most to least frequent, based on the intuition that more frequent classes would be easier to predict, and can be used to help predict the less frequent classes. Wang et al. [5] also tried other orderings and found no notable effects on the final results. Special START and END tokens were added to the beginning and end of the list of ordered labels, so that the RNN can know when to start and stop decoding. PAD tokens were added so that batches of labels are of the same length.

5. Experiments

5.1. Training

For gradient descent, the Adam optimizer [17] was used since it was recommended in lecture as a good default choice for most models. The default values of learning rate = 0.001, beta1 = 0.9, beta2 = 0.999, and epsilon = 1e-08 were initially used. Batch size of 32 was used since it was recommended as a good default choice in [15]. The LSTMs used hidden state size of 512, same as Wang et al.’s model [5].

Since the vocabulary size is only 83 for this task (80 classes + 3 special symbols), I hypothesized that a small embedding size would be enough. Pretrained GloVe vectors come in sizes 50, 100, 200, and 300. I chose 100 as the embedding size in order to compare randomly initializing the label embeddings vs initializing with pretrained GloVe vectors [20]. The GloVe vectors were trained on the Wikipedia and Gigaword corpus that contains 6 billion tokens. For classes that contain multiple words, they were treated as bag of words, and the average of the GloVe vectors of every word was used as the initial label embedding. For example, for the class "sports ball", the vectors for "sports" and "ball" were averaged together. The intuition behind using pretrained GloVe vectors is that the semantic relationships captured by GloVe are similar to the label dependencies that the label embeddings should capture. Experiments for every model were ran with both random initialization and GloVe initialization. The special symbols were always randomly initialized.

For random initialization, Xavier initialization [14] was used. Xavier initialization was also used for all the weight parameters and zero initialization was used for all the biases. Another detail to note is that for the CNN sequence to sequence models, experiments were ran with both shared and separate LSTM parameters for the encoder and decoder RNNs. For regularization, weight decay of 0.0005 and dropout of 0.5 was applied to the fully connected layers in VGG16. Each model was trained for 20 epochs. After every epoch, the train and validation precision/recall/f1 were evaluated, and the model parameters were saved to checkpoint. This essentially allows for early stopping because checkpoints from earlier epochs can be used as the final model.

After the initial set of experiments, I noticed that for all the models, the training loss decreased very rapidly for the first few epochs, then decreased very slowly for the remaining epochs, as shown in figure 5. Note that the initial loss of 4.41 is assuming that the model starts off by assigning equal probability to every label (1/83), resulting in cross entropy loss of $-\log(1/83) = 4.41$. The training loss plot shows that the learning rate is likely too high, therefore, experiments were ran with lower learning rates of 0.0005

and 0.0001 for all the models. Also from the initial set of experiments, I noticed that for the top performing models, the train vs validation f1 score showed that there was some overfitting, as can be seen in figure 6. Therefore, an additional dropout of 0.2 was added to the LSTM cells for these models, as suggested by [26].

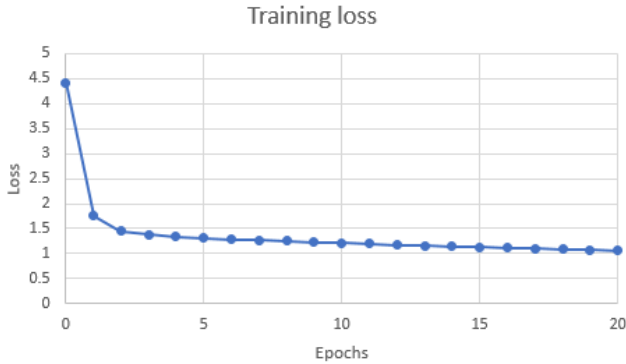


Figure 5. Training loss for CNN sequence to sequence

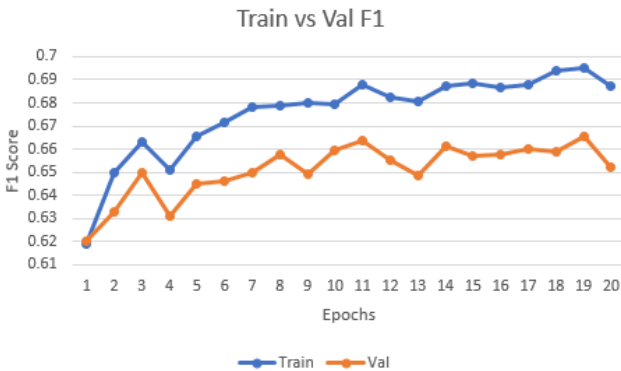


Figure 6. Train vs validation f1 score for CNN sequence to sequence

5.2. Results

The evaluation metrics used were precision, recall, and f1 score, averaged across all the evaluation samples. Precision is the percentage of predicted labels that is correct for a given sample. Recall is the percentage of correct labels that was predicted for a given sample. F1 score is a combination of precision and recall given by:

$$F_1 = 2 \frac{P \times R}{P + R} \quad (8)$$

Not that during evaluation, the order of the predicted labels does not matter. Table 3 shows the final test set results on MS-COCO for the various models.

Model	P	R	F1
WARP	59.8	61.4	60.7
Softmax	60.2	62.1	61.1
Binary cross-entropy	61.7	65.0	63.3
CNN as initialization w/ attn	69.6	59.3	64.1
Image-label joint embedding	66.6	65.3	65.9
Image-label joint embedding w/ attn	69.5	64.7	67.0
Wang et al.	69.2	66.4	67.8
CNN seq2seq w/ attn	70.9	68.0	69.4
CNN as initialization	71.5	67.4	69.4
CNN seq2seq	71.1	68.0	69.5

Table 3. Final results on MS-COCO

The first three models in gray are CNN only models where the top three classes with highest probabilities are outputted as the predicted labels. Their results are obtained from [5]. The model in yellow is the image-label joint embedding model implemented by Wang et al., and the results are their best numbers on MS-COCO [5]. These models were trained on 80k images and tested on 20k images from MS-COCO.

The models in green are the models described in the Methods section, trained on 20k images and tested on 5k images. The results shown are using the best configuration and hyperparameters for each model, determined by choosing the checkpoint with the highest validation f1 score. As can be seen from table 3, all these models outperformed the CNN only models, and three of them outperformed Wang et al.’s best model, producing state of the art results. The best performing model was the CNN sequence to sequence model. This model used learning rate of 0.0001, lstm dropout of 0.2, pretrained GloVe embeddings, and shared encoder/decoder weights. The CNN sequence to sequence with attention (learning rate 0.0001, no lstm dropout, separate encoder/decoder weights) and CNN as initialization models (learning rate 0.0001, no lstm dropout) also achieved very similar results.

Adding attention did not always help, which might be due to the fact that attention provides more benefit for longer sequences where it’s harder to maintain information from many time steps back. In these models, the input sequence is only five steps long.

5.3. Analysis

Since my hypothesis was that applying these NLP deep learning techniques would help with predicting multiple labels for an image, I analyzed the performance of the best model on images from the test set, categorized by their number of labels. The results are shown in figure 7.

The model did the best at classifying images with only one class, which is not surprising since those images are still the easiest to predict. The benefit of applying these

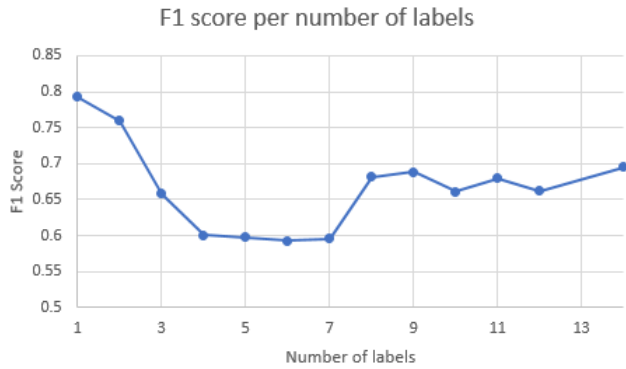


Figure 7. F1 score for different number of labels

NLP deep learning techniques can be seen in the model’s performance when there are greater numbers of labels. The performance decreases but plateaus and even rises a little as the number of labels increases, proving that the RNNs did a good job of capturing the relationships between labels.

I also analyzed the best model’s performance by class. The results are shown in figure 8 at the very end of this paper. Note that some of the classes have 0 f1 score, either because they did not appear in the test set, or they were actually never correctly predicted. The classes with the highest f1 scores tend to be visually distinct objects, such as zebra, person, tennis racket, and surfboard; which makes sense since objects with distinct image features should be easier to identify. Similarly, the classes with the lowest f1 scores tend to be objects that aren’t very visually distinct, such as handbag, backpack, tie, and bench; which makes sense since these objects can look similar to other objects.

Examining the model’s predictions for specific examples also show the benefit from applying these NLP deep learning techniques. Table 4 shows a few examples where the model made perfect predictions even though they have several labels. These examples show that the NLP techniques help with identifying objects that commonly appear together and allows for objects to be identified even when they’re visually barely discernible. For example, in the tennis picture, the tennis ball is very small and difficult to pick out from the background, but the model was still able to identify it; and in the restaurant picture, the knife and fork are almost completely hidden behind the bowl, but the model was still able to identify them. Table 5 shows a few examples where the model made completely wrong predictions. These examples show mistakes made due to the label relationships captured by the model; once an initial mistake is made, the label relationships lead to more mistakes. For example, in the teddy bear picture, the human sized teddy bear was likely mistakenly identified as a person, causing the model to also mistakenly predict car and bicycle.



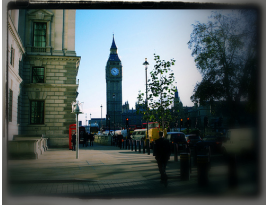

Image	Labels
	laptop, keyboard, mouse, tv
	person, tennis racket, sports ball, chair
	person, car, truck, traffic light, clock
	person, dining table, chair, bowl, fork, spoon, knife, cup, bottle, wine glass

Table 4. Examples from MS-COCO with perfect predictions

6. Future Work

An immediate improvement that can be made to the model is to perform beam search during inference and output the sequence with the best overall probability, instead of just outputting the class with the best probability at each time step individually. This should help for the mistake examples shown in table 5. For example, teddy bear might have had the second highest probability at the first decoding step, and its probability probably would have been higher than the sequence probability of the wrong predictions of person, car, and bicycle. More hyperparameter tuning should also result in further improvements. Due to time/resources constraints while trying several different models, only minor hyperparameter tuning was done on the learning rate and lstm dropout. Further tuning learning rate and regularization, as well as other parameters such as embedding size and hidden state size, can easily produce even better results. Training on the full dataset should also result in even more improvements. These models were only trained on a quarter of the full train set, and especially in deep learning, the training set size can have a huge impact on model performance. Finally, retraining the entire VGG instead of just the final layer might also result in improve-

Image	Labels	Predicted
	refrigerator, cat	sink, toilet
	oven, bottle, knife	cellphone, person
	teddy bear	person, car, bicycle
	vase, potted plant	bowl, dining table, cake

Table 5. Examples from MS-COCO with completely wrong predictions

ments, allowing the CNN part of these models to be better optimized for this task and dataset.

7. Conclusion

Despite all the possible steps for improvements detailed in the previous section, the current models already produce state of the art results on the multi-label image classification task on MS-COCO. Analyzing the results confirm the benefits of applying word embeddings, RNNs, and attention to this task, and specific examples show pretty remarkable and difficult predictions that were made possible by these NLP techniques.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. <https://arxiv.org/pdf/1409.0473.pdf>, 2016.

[2] C. G. et al. Pointing the Unknown Words. <https://arxiv.org/abs/1603.08148>, 2016.

[3] C. S. et al. Going deeper with convolutions. <https://arxiv.org/abs/1409.4842>, 2014.

[4] J. C. et al. Empirical evaluation of gated recurrent neural networks on sequence modeling. <https://arxiv.org/abs/1412.3555>, 2014.

[5] J. W. et al. Cnn-rnn: A unified framework for multi-label image classification. *CVPR*, 2016.

[6] K. H. et al. Deep residual learning for image recognition. <https://arxiv.org/abs/1512.03385>, 2015.

[7] K. X. et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. <https://arxiv.org/abs/1502.03044>, 2016.

[8] O. V. et al. Grammar as a Foreign Language. <https://arxiv.org/pdf/1412.7449.pdf>, 2015.

[9] O. V. et al. Pointer Networks. <https://arxiv.org/pdf/1506.03134.pdf>, 2017.

[10] T. M. et al. Distributed representations of words and phrases and their compositionality. *Neural Information Processing Systems*, 2013.

[11] T.-Y. L. et al. Microsoft COCO: Common Objects in Context. <https://arxiv.org/abs/1405.0312>, 2015.

[12] Y. Z. et al. Visual7W: Grounded Question Answering in Images. <https://arxiv.org/abs/1511.03416>, 2016.

[13] Z. A. et al. Label-Embedding for Image Classification. <https://arxiv.org/abs/1503.08677>, 2015.

[14] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. <http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf>, 2010.

[15] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.

[16] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[17] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. <https://arxiv.org/abs/1412.6980>, 2017.

[18] M. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *EMNLP*, 2014.

[19] O. Moindrot. <https://gist.github.com/omoindrot/dedc857cdc0e680dfb1be99762990c9c>.

[20] J. Pennington, R. Socher, and C. D. Manning. GloVe: Global Vectors for Word Representation. *EMNLP*, 2014.

[21] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. <https://arxiv.org/pdf/1409.1556.pdf>, 2015.

[22] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training Very Deep Networks. <https://arxiv.org/abs/1507.06228>, 2015.

[23] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to Sequence Learning with Neural Networks. <https://arxiv.org/pdf/1409.3215.pdf>, 2014.

[24] Tensorflow. <https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/slim/python/slim/nets/vgg.py>.

- [25] Tensorflow. https://github.com/tensorflow/models/blob/master/im2txt/im2txt/show_and_tell_model.py.
- [26] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent Neural Network Regularization. <https://arxiv.org/abs/1409.2329>, 2015.

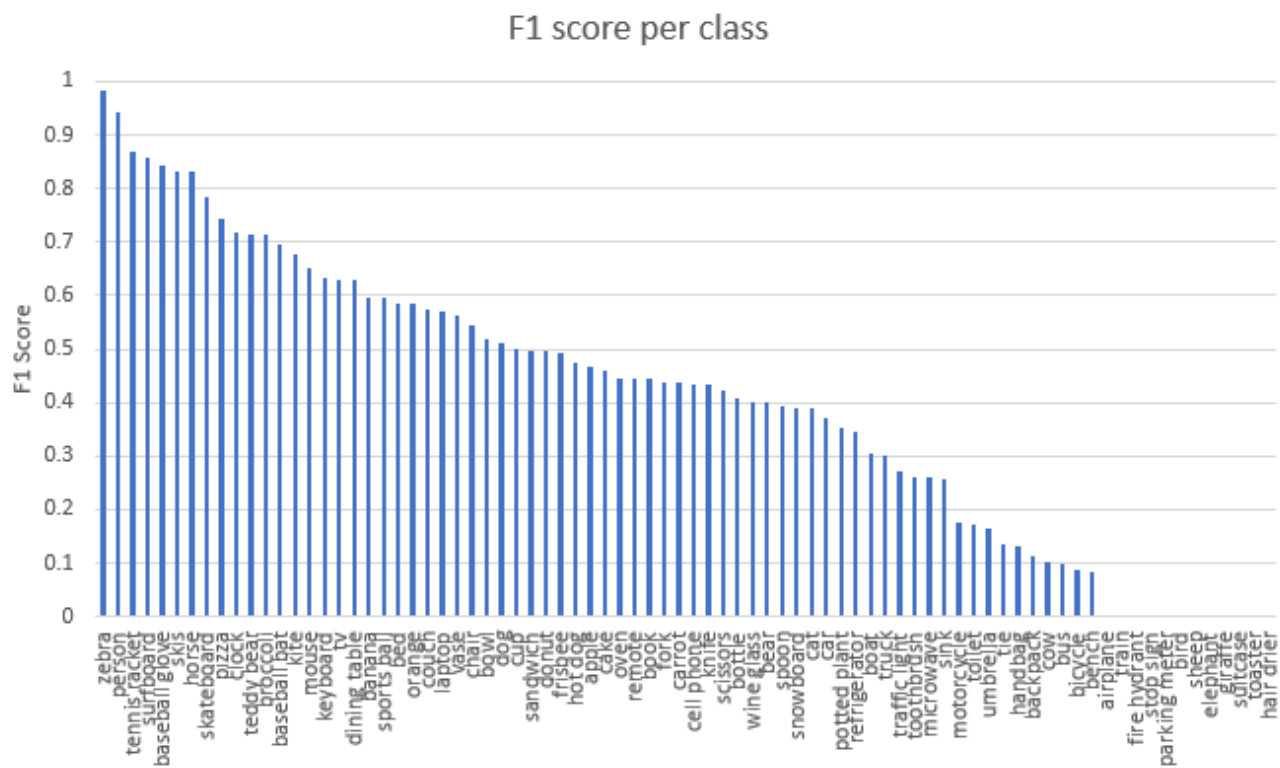


Figure 8. F1 score for different classes