Everything Looks Like Chicken (and Waffles)

A Neural Image Captioning System, built on Yelp image data. A submission to the Yelp Dataset Challenge.

Tamir Bennatan
timibennatan@gmail.com

***Abstract*— This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.**

I. INTRODUCTION

Here is an introduction. I will detail what an NIC system is, why it's hard and useful. I talk about how other researchers use specialized datasets for image captioning, and how in this work I used a real life dataset, where the annotaters are not given any instructions. I then talk about the work that I did, briefly outline experiements, results, and mention that I do qualitative error checking.

Here is an introduction. I will detail what an NIC system is, why it's hard and useful. I talk about how other researchers use specialized datasets for image captioning, and how in this work I used a real life dataset, where the annotaters are not given any instructions. I then talk about the work that I did, briefly outline experiements, results, and mention that I do qualitative error checking.

Here is an introduction. I will detail what an NIC system is, why it's hard and useful. I talk about how other researchers use specialized datasets for image captioning, and how in this work I used a real life dataset, where the annotaters are not given any instructions. I then talk about the work that I did, briefly outline experiements, results, and mention that I do qualitative error checking.

Here is an introduction. I will detail what an NIC system is, why it's hard and useful. I talk about how other researchers use specialized datasets for image captioning, and how in this work I used a real life dataset, where the annotaters are not given any instructions. I then talk about the work that I did, briefly outline experiements, results, and mention that I do qualitative error checking.

II. RELATED WORK

In recent years, the deep learning community has have achieved great success in core computer vision tasks, such as image classification, object identification, and image extraction (Sinha et al. 2018). These successes, coupled with advancements in neural machine translation and language modeling technologies, have enabled researchers to develop end-to-end neural network models that can be used for image captioning systems.

Vinyals et al (2014). describe such a system, which utilizes the power of Convolutional Neural Networks (CNN) for extracting complex visual features from images, and Recurrent Neural Networks (RNN) for generating new sequences. Inspired by the encoder-decoder framework for machine translation, the authors use a deep CNN to encode images to fixed-length vector representation, and feed this representation to an RNN which is used to generate a caption (further details in section III).

In their 2017 papers, Tanti et al. describe a set of architectures that fall under the general encoder-decoder framework. The primary difference between these architectures concerns how to feed the image features to the RNN layers, if at all. Tanti et al. rigorously test the efficacy of image captioning models with varying architectures on canonical image captioning datasets, in an attempt to deduce favorable architectures for image captioning models, and to form a conjecture as to what broad classes of tasks RNNs are best suited for.

Xu et al. (2016) achieve state of the art performance on standard datasets by incorporating an attention machanism. The authors argue that attention proves useful for the image captioning problem, as a network can learn to focus on the important features of an image, and ignore noisy features that result from cluttered images. Attention also provides additional interpretability to an image captioning model, as one can visualize what section of an image the attention is focused on at each time step.

Previous work focuses on the use of specialized datasets that were collected for the purpose of image captioning and object segmentation tasks, such as Flickr-8k (Rashtchian et al.; 2010), Flickr-30k (Young et al.; 2014), and MS-COCO (Lin et al.; 2014). These datasets were collected by giving human annotators detailed instructions on how to annotate each image, resulting in relativly clean, albeit synthetic datasets. This work uses a much more difficult, real-world dataset collected by Yelp. As users are given no instructions on how to annotate each image before uploading it to the site, this dataset makes it much more difficult to learn an effective image captioning network. In this work, I discuss some of these difficulties, and undesireable behaviors that

arise when training an image captioning network on such data. I also experiment with several architectures proposed by Tanti et al., in an attempt to study which architectures perform best under such circumstances.

III. THEORETICAL BACKGROUND

In this section, I discuss several abstract perspectives regarding the problem of Neural Image Captioning (NIC) which motivated many of the design choices and experiments that follow. First, I discuss how one can fit the problem of generating a caption of an image into the probablistic framework. Then, I briefly describe encoder-decoder models, and how the models I built are inspired by the use of the encoder-decoder framework in neural machine translation. Finally, I highlight an important design choice made when specifying the architecture of an NIC model, and the conceptual implication of this choice.

A. Training as Maximizing Caption Likelihood

Although much of the previous work on image and video captioning involves stitching together several independent systems, researchers are achieving increasing success building single neural networks for image captioning that are fully trainable by back-propogation. Learning the parameters of such models has a natural probablistic interpretation; the objective is to directly maximize (with respect to the model weights, θ) the joint probability of the predicted captions given input images, I, and model weights. For a dataset of N image/caption pairs, this can be modeled as[1]:

$$\theta_{\text{MLE}} = \arg\max_{\theta} \sum_{i=1}^{N} \log(P(S^{(i)}|I^{(i)};\theta) \tag{1}$$

Where $(S^{(i)}, I^{(i)})$ is the i^{th} caption/image pair, and each caption $S^{(i)}$ is a sequence of tokens $(S_1^{(i)}, S_1^{(i)}, ...S_k^{(i)})$ for a fixed maximum sequence length, k.

We may simplify this by the chain rule of probability:

$$\theta_{\text{MLE}} = \arg\max_{\theta} \sum_{i=1}^{N} \sum_{t=1}^{k} P(S_t^{(i)}|S_{t-1}^{(i)}...S_1^{(i)};\theta) \tag{2}$$

Equation (2) is the inspiration for the training schema I used to learn the weights, to be discussed further in section **SECTION HERE**.

B. Encoder-Decoder Framework for Image Captioning

Traditional sequence prediction tasks - such as time series forecasting and language modeling - involve using a varying length sequence of inputs (whose ordering usually have a temporal interpretation), and predicting a single output (usually at the next time step). Neural networks are well equipped to solve this type of "many-to-one" problem, as sequences of inputs may be coerced to a fixed number of time steps, so that the network does not need to account for varying length input or outputs.

[1]Under the assumption that the correct captions the images are mutually independent.

Another class of sequence prediction task are those with varying length sequences as both inputs and outputs. These so-called "many-to-many" or "sequence-to-sequence" problems are more difficult than many-to-one problems, as the network must learn to produce predictions of varying lenghts.

An important example of a sequence-to-sequence problem is that of statistical machine translation (SMT). SMT models attempt to take in a sequence of words in one language, and output a sequence of words in another language, all while preserving the meaning of the input sequence and the coherence of the output sequence.

An approach which has proven effective in sequence-to-sequence prediction problems, including machine translation, is called the "Encoder-Decoder" architecture. This architecture is comprised of two parts: the "encoder" takes in a varying length input sequence, and encodes it into a fixed-length vector representation. The "decoder" model decodes this vector representation into a varying-length sequence prediction.

Cho et al. (2014) use this framework to build a SMT system which translates sequences from French to English. They use LSTM recurrent neural networks to encode input sequences, and LSTMs to decode the encoded representations to form the predictions. Cho et al. show that the encoded representations of input sequences preserve syntactic and semantic structure - thus, this representation is often referred to as a "sequence embedding." Variants of encoder-decoder models have become the state of the art in machine translation; in fact, Google has adopted this approach in their Google Translate Service (Wu et al.; 2016).

Modern neural image captioning (NIC) systems, including those described in this paper, adopt this approach for the problem of image captioning. Image captioning is similar to machine translation in that the output is a varying length sequence of natural language, but differs in that the input is an image, rather than a sequence. Thus, instead of using RNN models such as LSTMs to encode the input, NIC systems typically use CNN models to form a vector representation from an image. Recurrent neural networks then use this representation to form an output sequence. Thus, NIC systems fall under the Encoder-Decoder framework, with CNNs acting as encoders and RNNs acting as decoders [figure 1].

The use of CNNs as encoders is justified by the success of CNNs in core computer vision tasks, such as image classification and segmentation. It is believed that CNNs succeed in these tasks becasue they can extract meaningful and complex features images effectively. Thus, just as LSTMs an encode the semantic and syntactic of an input sequence in SMT applications, it is believed that CNNs encode meaningful properties of an input image in NIC systems.

C. Merge and Inject Models

In a NIC system, CNN image features are combined with previously predicted tokens in a sequence to predict the next token in a sequence. The Encoder-Decoder framework includes a broad class of model architectures, and there are many ways to condition the sequence prediction on the image

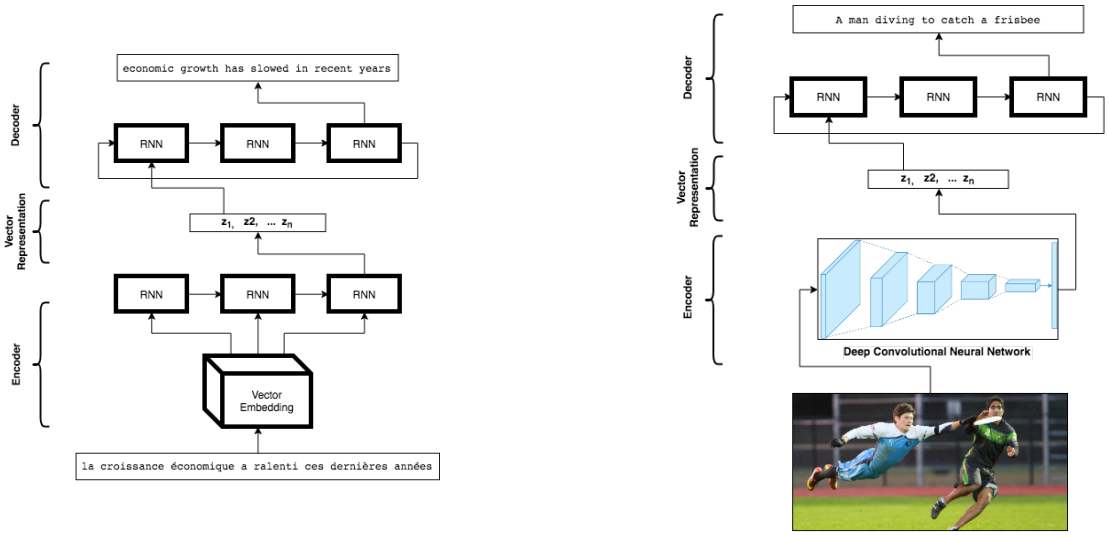

Fig. 1: The encoder-decoder framework in neural machine translation models (left) and image captioning models (right). In machine translation applications, an input sequence is encoded by an RNN to a vector represention, before decoded to an output sequence. Analogously, a neural image captioning system uses a CNN to encode an input image before generating an output sequence.

features, each of which uses the assigns a different role to the RNN component of a NIC system.

In their paper *What is the Role of Recurrent Neural Networks (RNNs) in an Image Caption Generator?*, Tanti et al. describe two important classes of NIC models that differ in their method of incorporating image features.

The first, referred to as "Condition-by-Inject" (or "Inject" models for short), incorporate the CNN image features directly to the RNN component. Using both the image features and the previously predicted tokens to predict the next token of the sequence. This is ususally done by conditioning the internal state of the RNN cells with the image representation, or treating the image representation as the first 'word' in the input sequence. The second method, called "Condition-by-Merge" (or "Merge" models for short), never incorporates the image features into the RNN model. Instead, the RNN is used to encode the linguistic features from previously predicted tokens, independent of the perceptual encoding formed by the CNN. Then, the linguistic and pereceptual encodings are merged to a single representation, from which the output prediction is made [figure 2].

The Inject and Merge architectures assign different conceptual roles to the RNN in a NIC system. In the Inject model, both perceptual and linguistic features are available to the RNN, which is responsible for generating the next predicted token. Thus, the RNN acts as a "generator" in the Inject architecture. In contrast, the RNN has no access to the image freatures in the Merge architecture. Instead, the RNN encodes an input sequence into a fixed length vector represenation, and a later fully connected layer is responsible for generation of new tokens. Thus, in the Inject architectures, the RNN can be thought of as an encoder, rather than a generator.

In the experiments described in section **ENTER SECTION**, I compare models using the Inject and Merge architectures. The idea that the choice of an Inject/Merge architecture dictates the 'role' of the RNN in a NIC adds adds a new persepctive to these experiments. The relative efficacy of one architecture over another not only shows us which architecture is more appropriate for this taks, but provides insights into what settings RNNs best suited in general.

IV. The Yelp Dataset

Yelp maintains a public dataset for the purposes of research and education. Part of this dataset is a collection of over 200,000 images uploaded by users, 100,807 with captions written by users upon uploading these images. This subset of images is what I used to train my image captioning system.

Compared to standard benchmark datasets for image captioning, such as Flickr-8k and Flickr-30k, the Yelp dataset makes for a much more difficult task in training an effective NIC model.

Standard datasets typically have 4-8 training captions per image, which helps elucidate which words are related the an image's content, and which words are a result of more noisy phenomena, such as an author's writing style.

More importantly, datasets such as Flickr-8k and Flickr-30k were collected by giving human subjects detailed instructions on how to form a training caption [figure 3]. In contrast Yelp users are given no guidance on what photos to upload to the site, nor how to write corresponding captions. The result is a dataset wit a much larger variablility in caption style, content and relevence to the task of image captioning. More severely, since the users are not given a common set of instructions on how to construct training captions, the task of modeling the image-caption relationship is not well defined.

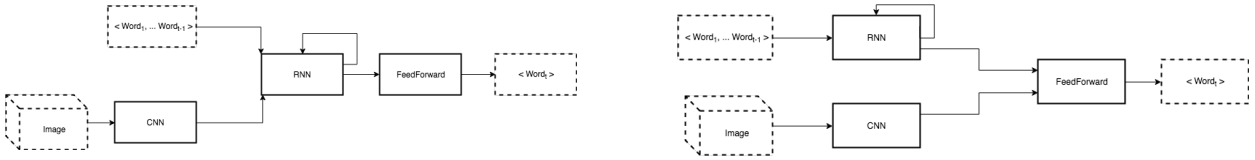

Fig. 2: A bare-bones diagram of the Inject architecture (left) and Merge architecture (right). The former conditions the RNN with both the image and linguistic features, and uses the RNN to generate new tokens. The uses the RNN to encode the previously predicted tokens into a vector representation, rather than to generate new tokens.

Fig. 3: Instructions presented to Amazon Turkers in the collection of the Flickr-8k dataset (Rashtchian et al., 2010). Image extracted directly from [7].

Figure 4 shows four images that make it difficult to train a NIC on the Yelp image dataset.

It's clear that for a NIC to perform well on the Yelp dataset, one would have to invest considerable effort in structuring and cleaning the training data. As I only did minimal preprocessing (see section **INSERT PREPROCESSING SECTION**, I do not expect my system to generalize well. Instead, my goal is to study the practical challenges of building an image captioning system on 'real-life' data. Perhaps the most pertinent challenge is collecting structured data which is designed for the task.

Fig. 4: Sample images from the Yelp image dataset, chosen to exemplify the varying sources of noise in the dataset.

- *Top-left:* Rarely encountered noun-phrases such as 'Artery clogger' and apocryphal words like 'Head-which' make it difficult to identify words tied to an image's content.
- *Top-right:* Captions which are unrelated to the image content.
- *bottom-left:* Wordy captions, which are likely to have content that is particular to an image, and does not generalize to images with similar content.
- *bottom-right:* Over simplified captions. This training example may cause a neural network to learn a link between the features of strawberry shortcake to the word *dessert*, which describes a much larger class of dishes.

V. METHODOLOGY

We started by splitting the labeled data into training and validation splits of sizes 40,000 and 10,000 examples, respectively. We kept these splits consistent in all experiments, so that we could gauge the relative effectiveness our different models. For each of the three algorithms we used, we chose which size heuristic(s) to use based on which heuristic(s) yielded the highest validation accuracy when used to select the largest digit during preprocessing. We also used this validation set to tune model-specific parameters.

A. Regularized Logistic Regression: Model Hyperparameters

The two hyperparamters we tuned the for logistic regression were the inverse-regularization coefficient, C[2], and the regularization loss function (either *l1* or *l2*). Using the processed datasets generated from using each of the four size heuristics, we ran a grid-search over 24 hyperparameter combinations, and selected the combination that yielded the best validation accuracy.

[2]In the notation seen in class, $C \propto 1/\lambda$.

B. Feedforward Neural Network: Model Hyperparameters

The two hyperparameters we tuned when using FFNNs were the learning rate, η, and the model architecture. We did not implement regularization.

C. Feedforward Neural Network: Learning Optimizations

We began by trying to fit our models using simple SGD, but quickly found that our models were not converging.

To address this, we implemented *momentum* to help calculate the ammount each weight θ should be updated on iteration t, denoted ν_t, where ν_t is defined:

$$\nu_t = \gamma\nu_{t-1} - \eta\nabla_\theta \text{Loss}(\theta)$$

Where γ is an additional hyperparameter introduced.

We also implemented *power scaling* - a method of slowly reducing the learning rate η with each epoch. At the end of each epoch, η is updated as follows:

$$\eta_{\text{new}} = \frac{\eta_{\text{old}}}{(\text{Epoch number})^p}$$

for a scaling parameter p. This allows the weights to change drastically in the begining of training, and slowly dampens the change in weights as training progresses, so that local maxima are not overlooked.

We found that introducing these two learning optimizations helped the SGD algorithm minimize loss much faster. Thus, for all experiments with FFNN, we used both momentum and power scaling, with the hyperparameters fixed at $\gamma = .9, p = .9$.

D. Convolutional Neural Network: Hyperparameter Tuning

We considered many hyperparameters when tuning our CNN models. The most important hyperparameters are summarized in Table 1.

We started by training all four architectures on the data preprocessed using Heuristic 1 with the high learning rate of .0001. After observing that architectures 2 and 4 yielded much higher validation accuracy than architectures 1 and 3, we decided to omit the latter architectures from all further experiments.

We then took a random sample of 64 images from EMNIST, and extracted the "largest" digit from each image using each of the four size heuristics defined. We observed by visual inspection that when using Heuristics 2 and 4, we isolated the digits corresponding to the labels more frequently than when we used Heuristics 1 and 3. We decided to use data preprocessed with Heuristics 2 and 4 in all further experiments.

We then performed a gridsearch on the remaining hyperparameter combinations permitted by the constrained hyperparameter ranges [Table 2], totaling 32 hyperparamter combinations.

E. CNN: Learning Optimization

We learned the weights of our networks using the *RMSProp* algorithm, with the hyperparameter ρ fixed at 0.9. While learning, we used an early stopping scheme, which caused training to stop if the validation accuracy has not improved for 8 consecutive epochs.

TABLE I: Hyperparameter Ranges Explored for CNN models

Hyperparameter	Range Explored
Model Architecture	{Architecture 1, Architecture 2, Architecture 3, Architecture 4}
Data Augmentation - (Batch size, Rotation Range)	{(1, [0, 0]), (8, [-10, 10]), (16, [-20, 20]), (16, [-30, 30])}
Learning Rate	{.00001, .00005, .0001}
Digit Size Heuristic	{Heuristic 1, Heuristic 2, Heuristic 3, Heuristic 4}

TABLE II: Constrained Hyperparameter Ranges

Hyperparameter	Range Explored
Model Architecture	{Architecture 2, Architecture 4}
Data Augmentation - (Batch size, Rotation Range)	{(1, [0, 0]), (8, [-10, 10]), (16, [-20, 20]), (16, [-30, 30])}
Learning Rate	{.00001, .00005}
Digit Size Heuristic	{Heuristic 2, Heuristic 4}

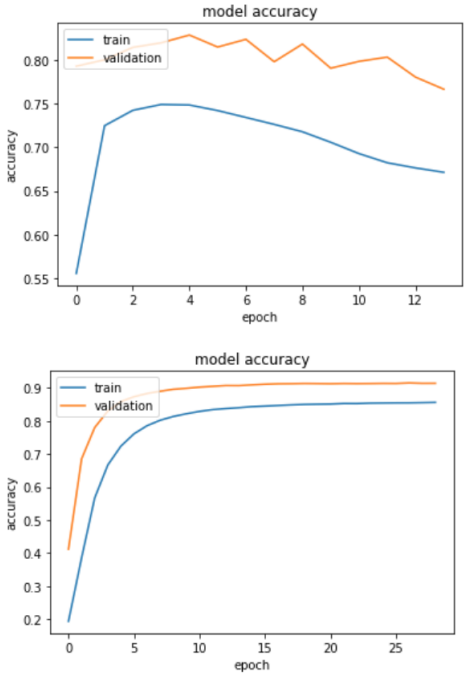

Fig. 5: Learning curves for training periods with $\eta = .0001$ and $\eta = .00005$, with early stopping. Note: training accuracy is lower than validation accuracy. This is because we used data augmentation, and the new randomly rotated images made the training data "harder" to learn than the validation data.

At first, we tried learning using learning rate of $\eta = .0001$, but we observed through the use of learning curves that the convergence had very high variance. After reducing the learning rate to .00005, we achieved smoother learning [Figure 7]. Thus, we limited the range of values or η to consider to $\eta \in \{.00005, .00001\}$.

F. CNN: Aggregated Predictions

We initialized the weights of our CNN's randomly. As such, every time we fit a CNN with fixed configuration, the final weights were not guaranteed to be the same, potentially leading to different predictions.

To account for this randomness, we fit two models of each hyperparameter configuration, each with different random initialization. This allowed us to keep the better of the two trained models (based on validation-set accuracy) for each hyperparameter configuration. Further, this presented us with the opportunity to aggregate the predictions of our best

performing CNN configurations[3].

We experimented with the aggregations of the predictions of our four best CNN models, in terms of validation accuracy. We refer to these CNN configurations as *CNN 1-4*.

Two of these aggregations increased validation accuracy by over 1%. We refer to these aggregations as *Aggregation 1* and *Aggregation 2*.

Aggregation 1:

- Predictions of four models aggregated.
- All four top CNN configurations used exactly once:
  - CNN 1: Architechture 2, Batch Bize = 16, Rotation Range [-.2, .2], η = .00005, Size Heuristic 2
  - CNN 2: Architechture 4, Batch Bize = 16, Rotation Range [-.2, .2], η = .00005, Size Heuristic 2
  - CNN 3: Architechture 2, Batch Bize = 16, Rotation Range [-.2, .2], η = .00005, Size Heuristic 4
  - CNN 4: Architechture 4, Batch Bize = 16, Rotation Range [-.2, .2], η = .00005, Size Heuristic 4

Aggregation 2:

- Predictions of four models aggregated.
- CNN 1 and CNN 2 were both trained twice with different intitializations, and included in the aggregation twice.

VI. RESULTS

In this section we discuss the hyperparameters which had the largest impact on the performance of each model, and then we compare the perforance of each of our tuned models on the validation set.

A. Regularized Logistic Regression

After doing a thorough search through many $\big(\text{C, Regularization Loss Function}\big)$ combinations, we found that the hyperparameter which had the largest impact on validation accuracy was the regularization loss function. The best model trained with $l1$-loss achieved a validation accuracy over 1% higher than the best model trained with $l2$-loss [Table 3].

TABLE III: Best Logistic Regression Models, Trained with l1/l2 Loss

Loss Function	Regularization Parameter	Validation Accuracy
l1-loss	0.09	0.731
l1-loss	.27	**0.735**
l2-loss	.01	0.723
l2-loss	.03	0.721

Models trained with $l1$-loss regularization find sparse solutions - meaning that many input features do not contribute to the decision boundary. This has an interesting interpretation for the EMNIST classification problem; it means that the Logistic Regression model learned that some pixel values (features) do not help to predict the largest digit, and should be ignored.

[3]To aggregate predictions of several models, we used the class which was predicted the most frequently amongst all models to be the aggregated class prediction.

For example, since we padded all images with a black border during preprocessing, these pixel values do not help to distinguish between the digits. Its reasonable to assume that the Logistic Regression model, trained with $l1$-loss learned to ignore the values of those pixels.

B. Feedforward Neural Network

We experimented with three model architectures when fitting our FFNN:

TABLE IV: FFNN Architectures

Architecture Number	Number of Hidden Layers	Hidden Layer Dimensions
Architecture 1	1	(128)
Architecture 2	2	(64, 20)
Architecture 3	2	(128, 64)

We trained all models for 20 epochs with SGD, using both momentum and power scaling.

After some experimentations, we found that the configuration $\eta = .01, \gamma = .9, p = .9$ led to a reasonable convergence rate. Using these hyperparameter configurations, we fit each of the architectures described in table 3, and got the following results:

TABLE V: Validation Accuracies for FFNN Experiments

Architecture Number	Validation Accuracy
Architecture 1	.55
Architecture 2	.64
Architecture 3	.64

Of the three architectures we used, those with two hidden layers achieved almost 10% better accuracy on the validation set. We have not tuned these models thoroughly, however, and so we cannot yet conclude that FFNNs with two hidden layers perform better on the EMNIST classification task than those with one hidden layer. Perhaps with more training, changes to the number of hidden units, and tuning to the learning hyperparameters γ and p, a FFNN with one hidden unit could outperform Architectures 2 and 3.

C. Convolutional Neural Network

When training our CNNs, the performance of our models was not very sensitive to the learning rate η. In fact, we saw no evidence that models trained with the learning rate $\eta = .00001$ yielded higher validation accuracy than models trained with $\eta = .00005$ - all else fixed.

A hyperparameter which had a noticeable effect was the choice of data augmentation scheme. Our best model trained without data augmentation yielded validation accuracy almost 2% lower than our best model trained with data augmentation.

We found that the benefits of data augmentation diminished as we increased the rotation range past $[-20, 20]$. One explanation for this is that if one distorts the training data too severely, it makes the learning problem *"too hard"*, and

TABLE VI: Best CNN Models for Different Data Augmentation Schemes

CNN Architecture	Digit Size Heuristic	Batch Size	Rotation Range	Validation Accuracy
4	2	1	-	89.4
4	2	16	[-20, 20]	**91.2**
4	2	16	[-30, 30]	90.3

a classifier has trouble picking up on patterns in the modified training data.

Thus, we found that the best configuration for data augmentation was a Batch Size of 16, with rotation range $[-20, 20]$ [table 6].

We also found that aggregating predictions of serveral models led to an increase in validation accuracy.

The four CNN configurations which achieved the best validation accuracy are specified in section *4.f*, and are referred to as *CNN 1-4*. The predictions of these models were pooled to create Aggregation 1. The pooled predictions of Aggregation 1 achieved better validation accuracy than any of the individual models used to construct it. Further, we found that Aggregation 2 yielded the best validation accuracy [Table 7].

This may suggest that the errors which of each of the models that we pooled "canceled out," simulating the predictions of a classifier with lower variance and increased prediction performance. Furthermore, due to the fact that all four models used to construct Aggregation 2 were trained on data preprocessed using Size Heuristic 2 - and that the predictions of Aggregation 2 led to the highest validation accuracy we achieved - we are inclined to believe that Heuristic 2 does the best job of identifying the "largest" digit in each image amongst the heuristics we proposed.

Overall, our CNN models had the performed the best out of all our models; our final predictions were those of Aggregation 2 - with a validation accuracy of 92.9% and a public leaderboard accuracy of 93.0%. This validates our initial hypothesis that CNNs are an appropriate model class for the EMNIST classification problem.

VII. Discussion and Conclusions

After running our experiments, we have concluded that CNNs achieve the best performance out of the three model classes that we trained. However, due to performance issues, we did not tune the hyperparameters of our FFNNs thoroughly enough to conclude that FFNN cannot achieve competitive performance. We would likely benefit from tuning our model architectures, and allowing our networks to train for more epochs.

Furthermore our FFNN implmentation was very simplistic. Some of the optimizations we could implement that would likely help performance are:

- Regularization, and tune regularization parameter. We could also try to limit overfitting by implementing dropout.
- Experiment with more learning methods (besides SGD), such as Batch Gradient Descent, and more advanced update rules, such as RMSProp or ADAM.
- Backpropogation optimizations, such as Batch Normalization.

Although we meticulously tuned the hyperparameters associated with our CNN models, we only considered a limited scope of model architectures. All four architectures we experimented with were variants of the LeNet architecture.

Although LeNet has proved effective in classifying handwritten digits by many authors, we would like to experiment with a more diverse range of architectures in the future. One particularly exciting recent development in deep learning are Capsule Networks, proposed by Hinton et al. (2017), which may be more robust to changes in image orientation and overlapping images than CNNs - and thus applicable to this problem.

Finally, we would like to invest more effort in improving our preprocessing methods. After visual inspection of 100 misclassified validation examples, we noticed that many of the digits that we extracted from these examples did not correspond to the correct label - meaning that we failed to extract the "largest" digit during preprocessing. We believe that our preprocessing is the bottleneck hindering us from building a more successful classifier, and that if we were to improve our preprocessing methods substantially then we could produce a more competitive solution.

TABLE VII: Highest performing CNN Models and Aggregations

CNN Configuration	Validation Accuracy
CNN 1	.910
CNN 2	.918
CNN 3	.906
CNN 4	.911

Aggregation	Validation Accuracy
Aggregation 1	.921
Aggregation 2	***.929***

Validation accuracies of best four CNN configurations, and those of aggregated predictions. Aggregation 1 was created by pooling the predicions of CNN 1-4 once, and Aggregation 2 was created by pooling the predictions of CNN 1 and 2 twice - after training both configurations twice with different initializations.

References

[1] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," *CoRR*, vol. abs/1411.4555, 2014. [Online]. Available: http://arxiv.org/abs/1411.4555

[2] M. Tanti, A. Gatt, and K. P. Camilleri, "Where to put the image in an image caption generator," *CoRR*, vol. abs/1703.09137, 2017. [Online]. Available: http://arxiv.org/abs/1703.09137

[3] ——, "What is the role of recurrent neural networks (rnns) in an image caption generator?" *CoRR*, vol. abs/1708.02043, 2017. [Online]. Available: http://arxiv.org/abs/1708.02043

[4] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[5] R. K. Sinha, R. Pandey, and R. Pattnaik, "Deep learning for computer vision tasks: A review," *CoRR*, vol. abs/1804.03928, 2018. [Online]. Available: http://arxiv.org/abs/1804.03928

[6] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," *CoRR*, vol. abs/1502.03044, 2015. [Online]. Available: http://arxiv.org/abs/1502.03044

[7] C. Rashtchian, P. Young, M. Hodosh, and J. Hockenmaier, "Collecting image annotations using amazon's mechanical turk," in *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, ser. CSLDAMT '10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 139–147. [Online]. Available: http://dl.acm.org/citation.cfm?id=1866696.1866717

[8] P. Young, A. Lai, M. Hodosh, and J. Hockenmaier, "From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions," *Transactions of the Association for Computational Linguistics*, vol. 2, pp. 67–78, 2014. [Online]. Available: https://transacl.org/ojs/index.php/tacl/article/view/229

[9] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[10] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," *CoRR*, vol. abs/1609.08144, 2016. [Online]. Available: http://arxiv.org/abs/1609.08144