Image Classification With EMNIST - An Extended Version of MNIST

Tamir Bennatan
tamir.bennatan@mail.mcgill.ca
260614526

Kyle Levy
kyle.levy@mail.mcgill.ca
260612028

Phillip Ryjanovsky
Philip.ryjanovsky@mail.mcgill.ca
260604024

I. INTRODUCTION

The MNIST dataset - comprising of images of handwritten digits of 10 classes - is a popular dataset, widely used as a benchmark for learning and classification tasks amongst the Neural Network and Computer Vision communities. MNIST enjoys several favorable charactaristics which have contributed to its widspread adoption (Cohen, et. al; 2017). First, the dataset is relatively small (each of the 60,000 examples are $(28x28)$ pixel grayscale images), which makes it feasible to work with the dataset on most processors [Figure 1]. Second, the digits have been centered within each image, have consistent heights and similiar widths. The digits are all distinguishable to the human eye, and the labels associated with each image are accurate. Because of these properties, many reserachers have achived near perfect accuracy in the image classification task on MNIST (for example: Lecun et. al; 2001).

In this project, we address the digit classification task on a more difficult dataset called Extended-MNIST (EMNIST). Each example int he EMNIST daset was generated with the following procedure:

1) Randomly select 2 or 3 images from MNSIT.
2) Rescale each image by a factor ranging from 40% to 120% the original size (the label of the generated image is that of the MNIST image which was scaled up by the largest factor).
3) Apply random rotations to each image.
4) Superimpose these rotated images onto a random background.

The inherent randomness in the EMNIST examples makes image classification on EMNIST more difficult than on MNIST [Figure 2].

In this report, we discuss the pre-processing measures we took to adress the noisy features of the EMNIST examples. We then summarize our process in fitting and tuning Logistic Regression, Feedforward Neural Network, and Convolutional Neural Network (CNN) classifiers. CNNs proved most effective; we achieved 92.6% validation accuracy using the aggregated predictions of four CNN models. We then summarize our results and our shortcomings, and ways in which we can improve our accuracy in the future.

II. FEATURE DESIGN

The EMNIST dataset introduces several difficulties in building an effective classifier. The random backgrounds of each example in the EMNIST dataset were generated independently of the examles' labels. Thus, they only add noise to each image, which can impede effective learning by a machine learning algorithm.

It is also unclear which digit in each image was scaled up the most during the construction of each example in EMNIST, as we do not have prior knowledge of the dimension sof the MNIST digits used to create each example in EMNIST. This makes it difficult to isolate the digit within each EMNIST example that corresponds with the target label.

Finally, an effective classifier for the EMNIST dataset must be robust to rotations in images, as the EMNIST datsset was built by rotating MNIST images randomly. As most machine learning algorithms do not encode images in a way that is robust to rotations, this adds diffulty to the classification task.

A. Removing Noisy Backgrounds

The first preprocessing step we took was to remove the backgrounds of each example in EMNIST. Luckily, the pixels in each example corresponding to a digit extracted from MNIST have a fixed integer value of 255 (corresponding to the color white). This made it possible to replace the random backgrounds with black backgrounds, by converting all non-white pixels to black pixel values [Figure 3].

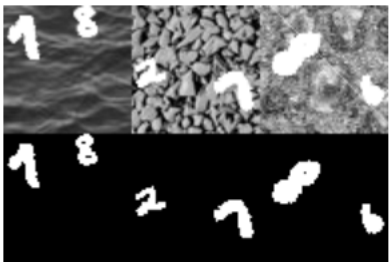

Fig. 3. Three sample images from EMNIST, before filling the backgrounds (top) and after filling the backgrounds (bottom).

B. Extracting the 'Largest' Digit

Each example in EMNIST contains several digits, but only one label. Intuitively, this makes the learning task difficult, as a machine learning algorithm will try associate the properties of several digits which an example's label, when in fact the task is to only correctly identify the *largest* digit.

Thus, we set out to extract the *"largest"* digit from each example in EMNIST. Since we do not know the original dimensions of each of the digits in each image, we employed a series of heuristics to define the size of an digit. These heuristics are:

Fig. 1. Three sample images from the MNIST dataset.

Fig. 2. Three sample images from the EMNIST dataset.

1) **Bounding Rectangle Area:** The area of the smallest upright rectangle that circumscribes the digit.
2) **Bounding Circle Area:** The area of the smallest circle which escribes the digit.
3) **Minimum Rotated Rectangle Area:** The area of the smallest (possibly rotated) rectangle that escribes the digit.
4) **Minimum Rotated Rectangle Maximum Dimension** The largest dimension (height,width) of the smallest (possibly rotated) rectangle that circumscribes the digit.

We refer to these huristics as *heuristics 1-4*, respectively.

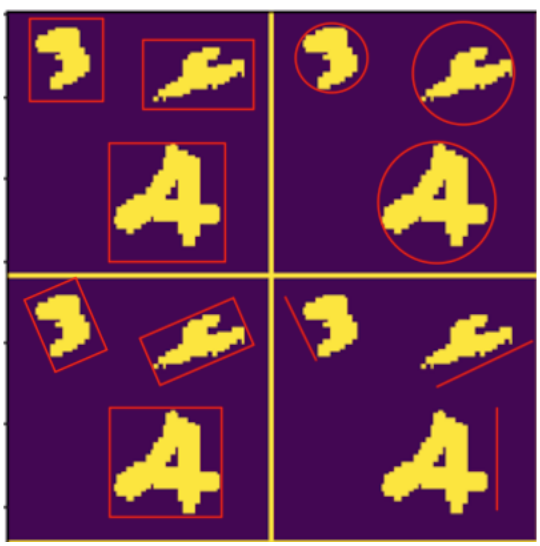

Fig. 4. A sample image from EMNIST, superimposed by the geometric objects used to measure the size of each image using heuristics 1-4. Heuristic 1 is the top left, Huerustic 2 is the top right, Heuristic 3 is the bottom left, and Heuristic 4 is the bottom right.

We treated the choice of which heuristic to use to measure the size of a digit as a hyperparameter, selected from them using cross-validation. Once we selected a heuristic, cropped each example to the a square which circumscribes the largest digit (according to this heuristic), and resized this square to 28 by 28 pixels.

C. Data Augmentation

To help make our model more robust to rotated digits, we augmented our training data with randomly rotated copies of each training example. Our rational is that if we train on randomly rotated images, our classifier will scale unseen randomly rotated images more effeciely. We tuned the number of times we rotated each image (called the *"batch size"*), and the range of degrees with which we rotated each image as hyperparameters.

III. ALGORITHMS

We trained classifiers of three types: Regularized Logistic Regression, Feedforward Neural Network, and Convolutional Neural Networks (CNN). We trained each of these classifiers on the datasets processed using each of size heuristics described above, though we only experimented with data augmentation when training the CNN.

Regularized Logisitc regression served as our benchmark, as it is a simple and easy to implement algorithm.

THIS IS FILLER. HERE WE TALK A BIT ABOUT IMPLEMENTATION AND WHAT WE TUNEDTHIS IS FILLER. HERE WE TALK A BIT ABOUT IMPLEMENTATION AND WHAT WE TUNEDTHIS IS FILLER. HERE WE TALK A BIT ABOUT IMPLEMENTATION AND WHAT WE TUNEDTHIS IS FILLER. HERE WE TALK A BIT ABOUT IMPLEMENTATION AND WHAT WE TUNEDTHIS IS FILLER. HERE WE TALK A BIT ABOUT IMPLEMENTATION AND WHAT WE TUNEDTHIS IS FILLER. HERE WE TALK A BIT ABOUT IMPLEMENTATION AND WHAT WE TUNEDTHIS IS FILLER. HERE WE TALK A BIT ABOUT IMPLEMENTATION AND WHAT WE TUNED.

Finally, we trained CNNs of four different archetecutres - each inspired by the LeNet archetecture by Lecun et. al (Legun et. al; 1995). These archetectures alternate between convolutional and max-pooling layers, and output predictions through two fully connected layers. One archetecture had three convolutional layers, two had four convolutional layers, and one had six convolutional layers.

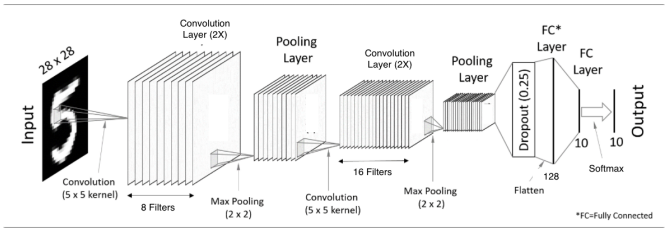

Fig. 5. One of the four CNN archetectures we experimented with. The other three differ slightly in the kernel size, pooling size, and number of convolutional layers. Image modified from [4].

A summary of the four architechtures we experimented with is presented in the apenndix. We refer to them as *Architecture 1-4*.

IV. 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 guage the relative effectiveness of the models. For each model, we used this validation set to select a digit size heuristic, based on which heuristic(s) yield the highest validation accuracy when used to select the largest digit. We also used this validation set to tune model-specific parameters.

A. Logistic Regression: Model Tuning

The two hyperparamters we tuned the for logistic regression were the inverse-regularization coefficient, C[1], 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.

B. Feedforward Neural Network: Model Tuning

THIS IS FILLER. HERE WE TALK A BIT ABOUT IMPLEMENTATION AND WHAT WE TUNEDTHIS IS FILLER. HERE WE TALK A BIT ABOUT IMPLEMENTATION AND WHAT WE TUNEDTHIS IS FILLER. HERE WE TALK A BIT ABOUT IMPLEMENTATION AND WHAT WE TUNEDTHIS IS FILLER. HERE WE TALK A BIT ABOUT IMPLEMENTATION AND WHAT WE TUNEDTHIS IS FILLER. HERE WE TALK A BIT ABOUT IMPLEMENTATION AND WHAT WE TUNEDTHIS IS FILLER.

C. CNN: 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 expiriments. We also decided to limit the range of the learning rate η to $\eta \in \{.000001, .00005\}$, as we saw that convergence was occuring with high variance.

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 3 and 4, we isolated the digits corresponding to the labels more frequently than when we used Heuristics 1 and 2. We decided to use data preprocessed with Heuristics 3 and 4 in all further experiemnts.

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

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

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	{.000001, .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	{.000001, .00005}
Digit Size Heuristic	{Heuristic 3, Heuristic 4}

D. 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.

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 6].

E. CNN: Aggregated Predictions

We initialized the weights of our CNN's randomly. As such, every time we fit a model with a certain configuration, the final weights may be different, leading to different predictions.

To account for this randomness, we fit two models of each hyperparameter configuration, each with different random

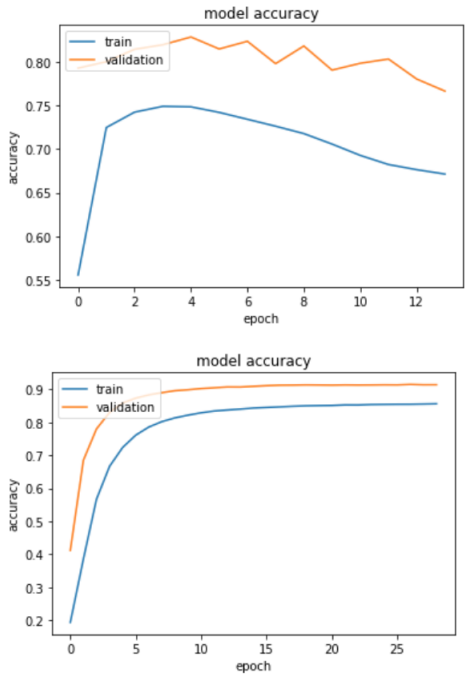

Fig. 6. Learning curves for training periods with $\eta = .0001$ and $\eta = .00005$, with early stopping. Note: training accuracy is lower than validation accuracy. This is because we used data augmentation, and the new randomly rotated images made the training data "harder" to learn than the validation data.

initialization. This allowed us to keep the better of the two trained models (based on validation-set accuracy) for each hyperparameter configuration.

Thus, for every combination of the hyperparameters (Data Augmentation Strategy, Size Heuristic), we had two models fit using Architecture 2, and two models fit using Architecture 4. This gave us the opportunity to aggregate predictions[2] across several trained models.

Two of the aggregations we experimented with increased validation accuracy by over 1%, which we considered substatial. We refer to these aggregations as *Aggregation 1* and *Aggregation 2*.

Aggregation 1:

- Predictions of four models aggregated.
- Two configurations - each repeated once with a different random initialization.
- Configuration 1: Architechture 2, Batch Bize = 16, Rotation Range [-.2, .2], η = .00005, Size Heuristic 3
- Configuration 2: Architechture 4, Batch Bize = 16, Rotation Range [-.2, .2], η = .00005, Size Heuristic 4
- Configuration 3: Architechture 2, Batch Bize = 16, Rotation Range [-.2, .2], η = .00005, Size Heuristic 3
- Configuration 4: Architechture 4, Batch Bize = 16, Rotation Range [-.2, .2], η = .00005, Size Heuristic 4

Aggregation 1:

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

- The same four configurations as in Aggregation 1, but each model configuration was trained and included twice.

F. Tf-Idf Features

Suppose we were allowed to compare only one word from each question in a pair to determine if the questions are duplicates, but had a choice of which words to compare. Which words should we choose? We would reasonably want to choose the most *important* word from each question, or the words which are most particular to each question.

For example, in the fabricated question pair:

1) *Who is the president of the United States?*
2) *What is the highest paying government position?*

We would probably derive more insight into the similarity/dissimilarity of these questions by analyzing the words *president* and *government* than if we compared more commonly occurring words, like *who* and *highest*.

We capture this intuition by creating a set of features that incorporate the Tf-Idf scores of each word. We used Scikit-Learn's `TfidfTransformer` class (Pedregosa et al.; 2011) to compute the Tf-Idf of a word w in document d using the formula:

$$Tf\text{-}Idf(w,d) = Tf(w,d) * Idf(w) \tag{2}$$

Where $Tf(w,d)$ is the number of times word w appers in document d (term frequency), and $Idf(w)$ is defined:

$$Idf(w) = \log \frac{1+n_d}{1+df(w)} + 1 \tag{3}$$

Where n_d is the number of documents[3] in the corpus, and $df(w)$ is the number of documents in the corpus that contain the word w.

The Tf-Idf weight of a word in a document will be high if: 1) the word appears many times in that document, and 2) the word has low document frequency. It serves as a natural heuristic for modelling the relative importance of a word within a document. Tf-Idf weights are useful for many NLP tasks; in fact, variations of this scoring scheme are used as term weights in nearly all vector space information retrieval models (Jurafsky & Martin; 2012).

Thus, we used Tf-Idf scores to craft four new features. The first two attempt to measure similarity of the "most important" word of each question in a pair. To do so, we find the word with the highest Tf-Idf weight in each question, then take the fastText word embedding for each of these words, and compute the distance between these vectors. We used Euclidean distance for one feature and Cosine distance for the second.

$$Feature_1 = ||Embed(w_1^*) - Embed(w_2^*)||_2 \tag{4}$$

$$Feature_2 = \frac{\langle Embed(w_1^*), Embed(w_2^*)\rangle}{||Embed(w_1^*)||_2 ||Embed(w_2^*)||_2} \tag{5}$$

[3]*Documents* and *texts* are questions, in this context.

Where

$$w_i^* = \arg\max_{w_i \in d_i} \{Tf\text{-}Idf(w_i, d_i)\} \qquad i \in \{1, 2\}$$

The third and fourth features measure the total Tf-Idf weight of the words two questions have in common, and the total weight of the words that they don't:

$$Feature_3 = \sum_{w \in \{d_1 \cap d_2\}} Tf\text{-}Idf(w, \{d_1 \cap d_2\}) \quad (6)$$

$$Feature_4 = \sum_{w \in \{d_1 \triangle d_2\}} \sum_{i=1}^{2} Tf\text{-}Idf(w, d_i) \mathbb{1}(w \in d_i) \quad (7)$$

G. Question Classifcation Features

IR-based question answering (QA) systems attempt to find short segments of text that answer a user's question by querying a database or searching the web (Jurafsky & Martin; 2017). These systems tend to perform better if they first constrain the answer candidates using a semantic categorization of the expected answer of an input question (Li & Roth; 2002).

For example, when considering the question:

Who is the president of the United States?

A QA system should limit its search to words that correspond to humans, as opposed to considering all noun-phrases.

Using the Text Retrieval Conference (TREC) question dataset (Voorhees; 2002), Roth and Li (2005) developed a hierarchical taxonomy for classifying the answer type of a question. This taxonomy consists of 6 coarse categories, and 50 fine sub-categories. They then annotated the 6000 questions in the TREC dataset using this taxonomy. The coarse answer categories are LOCATION, DESCRIPTION, ENTITY, NUMERIC, HUMAN, and ABBREVIATION, and some examples of fine categories are NUMERIC::Date and NUMERIC::Money. The task of predicting the answer type of a question is broadly termed *Question Classification*.

We hypothesized that knowing the answer types of the questions in the Quora dataset would be useful for the question duplicate detection problem, since two questions are more likely to have duplicate intent if they share the same answer type. Our next feature set tests this hypothesis.

We trained 5 LSTM models on the TREC dataset to predict the coarse categories of each question. These models each have slightly different architectures, but they all consisted of three layers:

1) Embedding layer using pretrained fastText 300-dimensional embeddings
2) LSTM layer of output dimension 50
3) Output layer (of varying types) of output dimension 6.

The details of the different architectures are in Table 3. Note that we trained these LSTMs on the entire TREC dataset without the use of the development set, so we make no statement about the efficacy of these models for question classification.

TABLE III
ARCHITECTURE DETAILS OF LSTM QUESTION CLASSIFIERS

Model	Output Layer	Dropout	Recurrent Dropout	Embeddings Trainable?	Input Length
LSTM 1	Dense	20%	-	NO	10
LSTM 2	Dense	-	20%	NO	10
LSTM 3	LSTM	-	20%	NO	10
LSTM 4	LSTM	20%	20%	YES	10
LSTM 5	LSTM	-	20%	NO	6

Using these models, we predicted the answer type of the questions in the the Quora dataset. Then, considering each LSTM model separately, we added these predictions to our baseline features, and measured the increase in our baseline XGB model using 3-fold cross validation. We also tried an ensembled approach, where the predicted class of each question is the class most predicted amongst the 5 LSTMs.

We found that the ensembled predictions, as well as a binary feature which indicates if these predictions coincide, increased the performance of our baseline XGB model the most (in terms of reduction of log-loss). Thus, these are the features in the second feature set which we propose:

$$Feature_5 = \{\text{Question 1 Answer Type Prediction}\}$$

$$Feature_6 = \{\text{Question 2 Answer Type Prediction}\}$$

$$Feature_7 = \mathbb{1}(\text{Answer Type Predictions Agree?})$$

H. Sentence Semantic Similarity Scores

Our final feature set uses a framework for measuring the semantic similarity of short texts proposed by Mihalcea et al (2006). This framework models the similarity of two texts as a function of the similarities of their component words and the specificity of each word. Word-to-word similarities are measured with respect to a semantic knowledge base (e.g. WordNet.)

Given a word-to-word similarity metric, $Sim(w_1, w_2)$, and two texts[3] T_1, T_2, the semantic similarity score of these two texts is computed by:

1) For each word $w \in T_1$, find the word $w^* \in T_2$ which maximizes $Sim(w, w^*)$.
2) Apply this same process to determine the words in T_1 that have the highest similarity to the words in T_2.
3) Sum up the similarities (weighted by a measure of specificity, such as Idf (3)), and normalize for the length of the sequences.

Note that words in one text are only compared to words in the other text *of the same part of speech*, as many knowledge-based similarity measures cannot be applied across parts of speech (Mihalcea et al; 2006).

$$SimText(T_1, T_2) = \quad (8)$$
$$\frac{1}{2}\Big(\frac{\sum_{w \in T_1} \max_{w^* \in T_2} Sim(w, w^*) Idf(w)}{\sum_{w \in T_1} Idf(w)} + \frac{\sum_{w \in T_2} \max_{w^* \in T_1} Sim(w, w^*) Idf(w)}{\sum_{w \in T_2} Idf(w)}\Big)$$

This framework guarantees the convenient properties of symmetry: $SimText(T_1, T_2) = SimText(T_2, T_1)$, and that the range of the text similarity scores is the same as that of the word-to-word similarity measure $Sim(w_1, w_2)$ used.

We used the python NLTK package for part-of-speech tagging (Bird et al; 2009). We implemented this framework using three similarity metrics available via NLTK's WordNet interface - path similarity, Leacock-Chodorow similarity (Leacock & Chodorow; 1998) and Wu-Palmer similarity (Wu & Palmer; 1994).

We also implemented a simplified version of this framework by removing the dependency on Idf scores in (8):

$$SimText_{simplified}(T_1, T_2) = \frac{1}{2}\Big(\frac{\sum_{w \in T_1} \max_{w^* \in T_2} Sim(w, w^*)}{|T_1|} + \frac{\sum_{w \in T_2} \max_{w^* \in T_1} Sim(w, w^*)}{|T_2|}\Big) \quad (9)$$

We made this simplification to reduce the correlation between these similarity scores and the Tf-Idf features proposed in section 4.B - which also depend on Idf scores.

We also extended this model by using a word-to-word similarity scores based neural word embeddings. Using pretrained GloVe word embeddings (Pennington et al.; 2014) available through the SpaCy Python package, we used the cosine similarity (5) of the embeddings of two words as the similarity function $Sim(w_1, w_2)$ in (8) and (9).

Using 3-fold cross validation, we added different subsets of these text-to-text similarity scores to our baseline feature set, and studied resulting boost in our baseline XGB model performance. Since the text-to-text scores are highly correlated when using path similarity, Leacock-Chodorow similarity and Wu-Palmer similarity as $Sim(w_1, w_2)$, we only tested one of these scoring functions at a time.

We found that including two semantic similarity scores, calculated using the simplified scheme (9) and using Leacock-Chodorow distance and embedding Cosine distance as $Sim(w_1, w_2)$, resulted in the largest increase in performance in our baseline model. These two similarity scores define the final feature set we propose.

$$Feature_8 = \{SimText_{Simplified}(T_1, T_2) \quad (10)$$
where $Sim(w_1, w_2) \equiv$ Leacock-Chodorow Similarity$\}$

$$Feature_9 = \{SimText_{Simplified}(T_1, T_2) \quad (11)$$
where $Sim(w_1, w_2) \equiv$ Cosine Distance (GloVe Embeddings)$\}$

V. RESULTS

To measure the importance of each of these feature sets, we trained logistic regression and XGB models on all the features described above. We then removed different subsets of Tf-Idf, question classification and semantic scoring feature sets, and re-tuned each model's hyperparameters using 3-fold cross validation. We then used these models to predict test-set labels. The prediction accuracy for the resulting models are summarized in Table 4, as well as those of our baseline classifiers described in section 4.A. We limit our analysis to the XGB model, as this performed much better than logistic regression.

Including all three feature sets results in the best performance, with a test set accuracy of 79.25% - 4.69% better than the baseline XGB model. Amongst the three feature sets we propose, removing the Tf-Idf features (4.B) results in the largest decrease in prediction accuracy relative to the full model - with a 3.01% decrease. Removing the question classification features from 4.C results in the smallest decrease in prediction accuracy, indicating that these features are the least meaningful of those we propose.

When using ensembled tree models such as XGB, one can use the number of times a variable is split upon as a measure of that variable's importance. This measure is called the *F-Score*. By plotting the F-Scores of each feature, we can gain insight into the importance of each variable used in a model, relative to the other variables present (Gareth et al; 2017).

A variable importance chart of the full model corroborates the result that the Tf-Idf features are the most meaningful, as these features are amongst those with the highest F-Scores [figure 1]. Semantic similarity scores also have high F-Scores. Interestingly, when the Tf-Idf features are removed, the semantic similarity scores become much more important relative to the remaining features, evident in Figure 2. This suggests that the Tf-Idf features and semantic similarity scores explain some of the same variance, and once the Tf-Idf features are removed the model must rely more heavily on the semantic similarity scores. Indeed, the features in these to feature sets are strongly correlated; the correlation between Feature 9 and Features 6 and 7 are 68.28% and -73.91%, respectively.

The variable importance charts also suggest that the question classification features are not meaningful. In fact, these three features are amongst four features with the lowest F-Score.

VI. DISCUSSION AND CONCLUSIONS

Using the three sets of features described in this paper, we were able to improve the accuracy of our baseline model from 74.56% to 79.25%. Of the three feature sets, those computed using Tf-Idf scores were the most impactful. This confirms our hypothesis that measures of word importance and specificity are useful in comparing the semantic similarity between two questions.

We also applied and extended a framework for modelling semantic similarities between short texts using knowledge-based word similarity scores, proposed by Mihalcea et al (2006). We found that these scores were useful in our classification task. These features were highly correlated with those calculated using Tf-Idf scores, so it may be that they are merely capturing the specificity of the words in each question, rather than the semantic similarity of these words.

TABLE IV

TEST SET ACCURACY SCORES, WITH DIFFERENT FEATURE SETS OMITTED

XGB AND LOGISTIC REGRESSION, WITH FEATURE SETS OMITTED								
Feature sets ommited	∅	{1}	{2}	{3}	{1,2}	{1,3}	{2,3}	{1,2,3}
XGB Test Set Accuracy	**.7925**	.7624 (.0301)	.7861 (.0064)	.7825 (.01)	.7585 (.034)	.7504 (.0421)	.7794 (.0131)	.7456 (.0469)
Logistic Regression Test Set Accuracy	**.6962**	.6838 (.0124)	.6951 (.0011)	.6837 (.0125)	.6836 (.0126)	.6743 (.0219)	.6827 (.0125)	.6728 (.0234)
BASELINE MODELS								
Manhattan LSTM	.7588	Human (400 Responses)			.8175	Random Guessing	& .5015	

Here, {1} are the Tf-Idf features from 4.B, {2} are the question classification features from 4.C, and {3} are the semantic similarity scores from 4.D. Cells contain each model's prediction accuracy, as well as the difference between prediction accuracy and the best model's prediction accuracy.

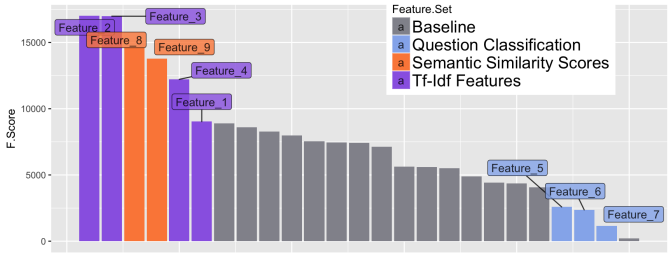

Fig. 7. Variable importance chart of model trained on all features.

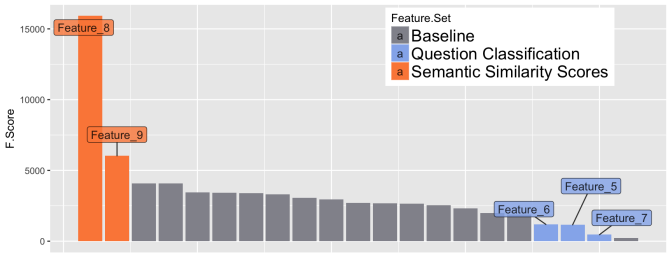

Fig. 8. Variable importance chart of model that excludes the Tf-Idf features.

Finally, we found that our answer type predictions were not useful for detecting duplicate questions. We suspect that this is because we used overly complex models to predict the answer type of a question. The TREC dataset has only 6,000 observations, but the number of parameters in our LSTM classifiers ranged from 70,506 to 28,141,268. It's probable that these classifiers were overfit to the TREC dataset, and did not generalize to the Quora dataset.

In future work, we would like to try using simpler models for predicting a question's answer type - such as Naïve Bayes or a Support Vector Machine. We could also try to incorporate predictions of the fine answer categories in the annotated TREC dataset, as apposed to the coarse categories we used.

We also think that an interesting extension to this work would be to extract features that concern the grammatical functions of the words in each question. For example, one could construct a dependency parse tree for each question, and compare the subjects and direct objects in each question, as well as the head words of their repsective parses.

VII. STATEMENT OF CONTRIBUTION

Tamir engineered the features and classifiers described, worked on the paper, and conducted model experiments. Jack contributed to the classifiers and paper, and experimented with advanced techniques for feature extraction, such as named entity recognition and headword extraction.

VIII. APPENDIX

REFERENCES

[1] Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved from http://arxiv.org/abs/1702.05373 (link is external)

[2] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition, Intelligent Signal Processing, 306-351, IEEE Press, 2001,

[3] Y. LeCun, L. D. Jackel, L. Bottou, A. Brunot, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard and V. Vapnik: Comparison of learning algorithms for handwritten digit recognition, in Fogelman, F. and Gallinari, P. (Eds), International Conference on Artificial Neural Networks, 53-60, EC2 & Cie, Paris, 1995.

BLOG POSTS

[4] Theart, R. (2017, November 29). Getting started with PyTorch for Deep Learning (Part 3: Neural Network basics). Retrieved from https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/

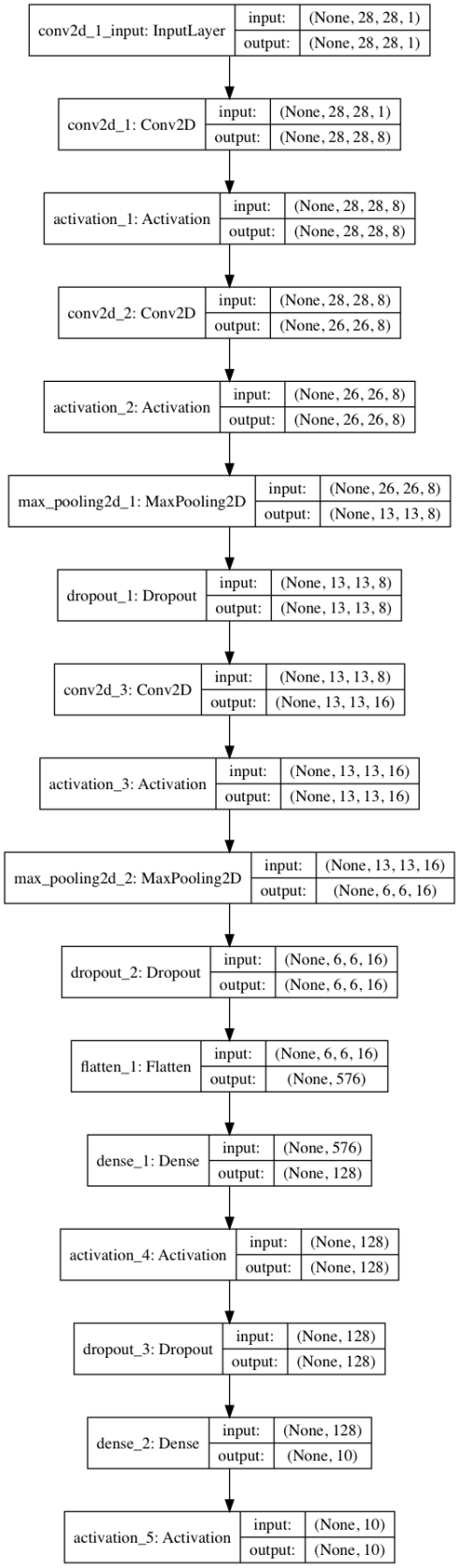

Fig. 9. Architecture 1: Three convolutional layers, 76,978 trainable parameters.

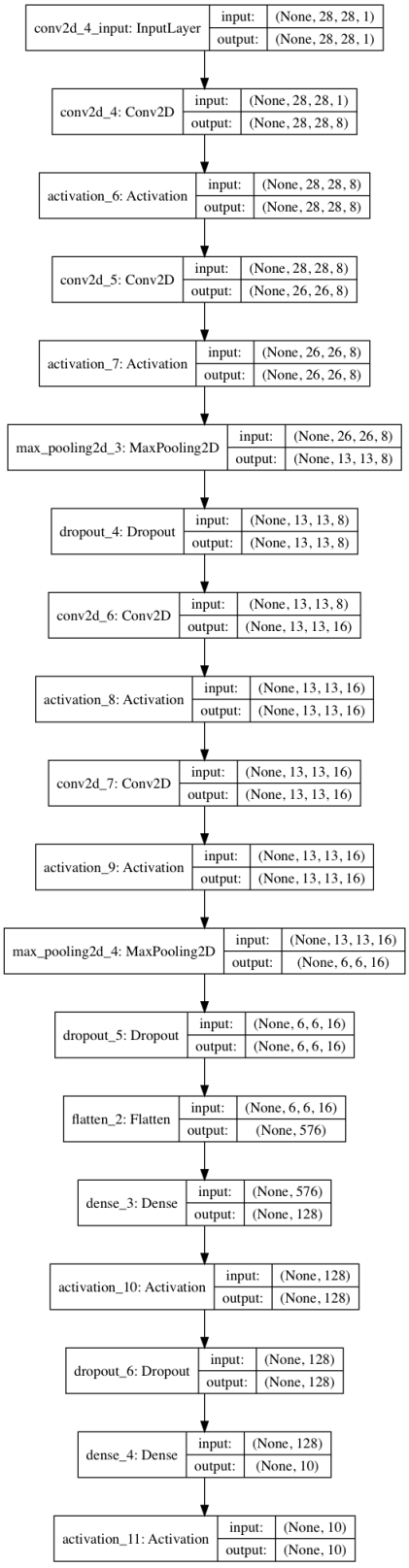

Fig. 10. Architecture 2: Four convolutional layers, 79,298 trainable parameters.

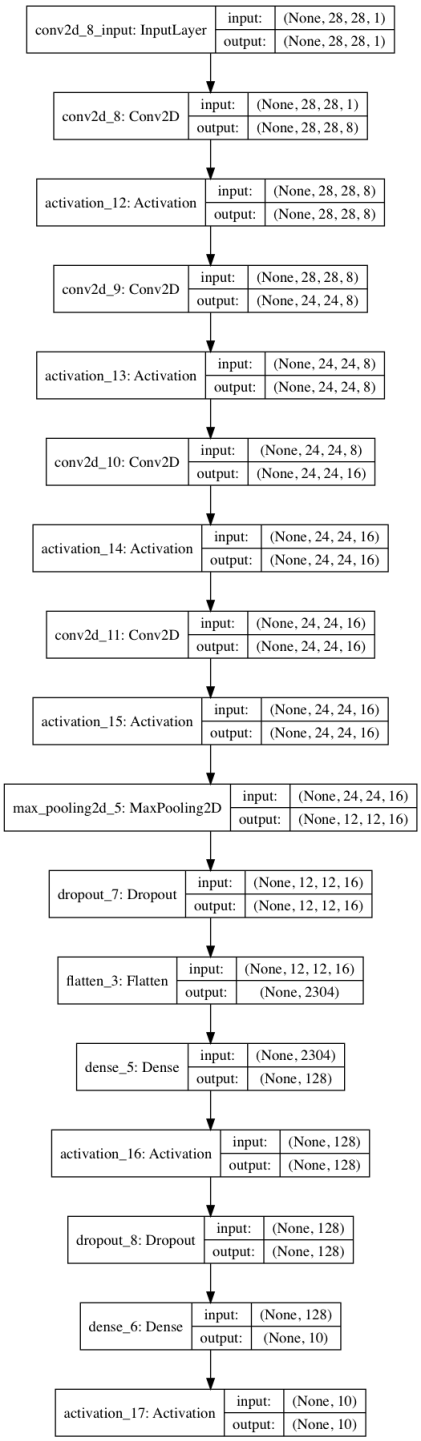

Fig. 11. Architecture 3: Four convolutional layers, 307,778 trainable parameters.

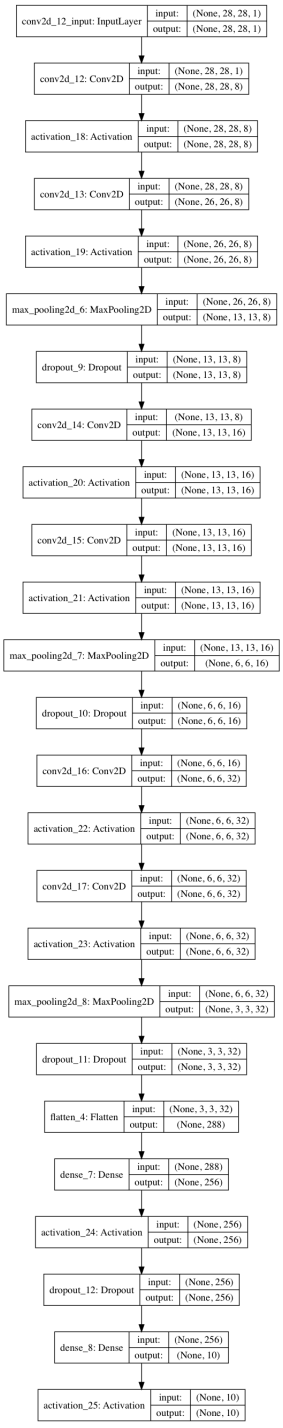

Fig. 12. Architecture 4: Six convolutional layers, 94,594 trainable parameters.