Mockingbird at the SIGTYP 2022 Shared Task: Two Types of Models for the Prediction of Cognate Reflexes

Christo Kirov
Google Research, US
ckirov@google.com

Richard Sproat
Google Research, Japan
rws@google.com

Alexander Gutkin
Google Research, UK
agutkin@google.com

Abstract

The SIGTYP 2022 shared task concerns the problem of word reflex generation in a target language, given cognate words from a subset of related languages. We present two systems to tackle this problem, covering two very different modeling approaches. The first model extends transformer-based encoder-decoder sequence-to-sequence modeling, by encoding all available input cognates in parallel, and having the decoder attend to the resulting joint representation during inference. The second approach takes inspiration from the field of image restoration, where models are tasked with recovering pixels in an image that have been masked out. For reflex generation, the missing reflexes are treated as "masked pixels" in an "image" which is a representation of an entire cognate set across a language family. As in the image restoration case, cognate restoration is performed with a convolutional network.

1 Introduction

The cognate reflex prediction task can be understood by considering a simple example. English *dream* is *droom* in Dutch and *Traum* in German. English *stream* is *stroom* in Dutch and *Strom* in German (so we can see that there is a bit of variation in how the cognate forms are realized). But now consider English *tree*, which is *boom* in Dutch and *Baum* in German. If there were a cognate in English, what would it look like? On analogy in particular with the *dream* example, one would expect the form to be *beam*—which is in fact cognate with the Dutch and German forms, though the meaning has shifted.

This study describes the two particular approaches to cognate reflex prediction code-named Mockingbird.[1] Both approaches use popular machine learning techniques adapted to the cognate reflex prediction task.

[1]The open-source implementation of both models: https://github.com/google-research/google-research/tree/master/cognate_inpaint_neighbors

The *transformer-based* approach popularized by Vaswani et al. (2017) is a particular instance of sequence-to-sequence (Seq2Seq) recurrent neural bipartite encoder-decoder architecture (Cho et al., 2014; Sutskever et al., 2014) equipped with multi-head attention mechanism. It has the advantages inherent to Seq2Seq models. In particular it can represent arbitrarily long-distance contextual dependencies both within and across words. This rich representational capacity comes at the cost of high model complexity and need for computational resources (Strubell et al., 2019; Liu et al., 2019; Brown et al., 2020). Our particular transformer-based model was originally developed for place name pronunciation (Jones et al., 2022), and models cognate sets as a *neighborhood* where we are trying to predict the pronunciation of a target feature given its neighbors.

The *image inpainting model* (Liu et al., 2018) relies on a simple convolutional neural network, or CNN (O'Shea and Nash, 2015), which trains fairly quickly even on CPU. It treats cognate sets as holistic objects ("images"), with individual convolution filters representing partial joint alignments between all languages at once. Unlike transformers or RNN-based models, however, convolutional models with finite kernel sizes cannot capture arbitrary amount of context. As an extreme example, it would be difficult for a convolutional model to learn that the first character in one language cognate should always be aligned with the last character in another language's cognate, especially if long words are possible.

2 Related Work

The establishment of cognate correspondences and the prediction of cognate forms has a long and venerable history that dates at least to the original work of William Jones that established the Indo-

European language family (Jones, 1786), and later the work of the Neo-Grammarians (Paul, 1880).

Prior computational work related to this problem includes early work by Covington (1996) and Kondrak (2000) on methods for aligning cognate pairs at the segment level, and more recently work specifically on the prediction of cognate forms (List, 2019a; Meloni et al., 2021). The work of Meloni et al. (2021) in particular is most similar to the current proposed methods in that they use a neural character-level encoder-decoder sequence-to-sequence model, and demonstrate that this shows good performance on the task of proto-form reconstruction for a fairly large dataset of Romance languages.

3 Neighbors Transformer Model

3.1 General architecture

The "neighbors" transformer model was originally developed for detecting inconsistencies in the readings of Japanese place names in an industrial-scale maps database (Jones et al., 2022). Japanese place names are notoriously difficult to read even for native speakers, since two different place names that happen to be written with the same kanji sequence may have quite different pronunciations. A famous example is 日本橋, which in Tokyo is read as *Nihonbashi*, but where the identically written name in Osaka is read as *Nipponbashi*. While the place names themselves are unlikely to be wrong in the data, there are many named features in the data set—buildings, establishments, and so forth—which are named after the area—e.g. an apartment building named *Mezon (= Maison) Nipponbashi*, where the reading may be in error. The neighbors model is designed to detect such errors by considering the reading of features in the area and detecting if there is an apparent inconsistency. In training the model is provided with the target feature's written form and its reading (in *hiragana*), and the written forms and readings of features that are neighbors within a small geographic distance from the target feature. Since inconsistencies in such neighborhoods involve the minority of cases, the model learns that the same spelled name within a neighborhood usually agrees in terms of the reading. The general architecture of the model as applied to Japanese place names is shown in Figure 1.

The cognate reconstruction problem is similar in spirit to the place name problem just described. In this case the "neighbor pronunciations" are replaced with the cognates in the set, with the "main feature" being the cognate form to be predicted. The spellings on the other hand are replaced by a string representing the name of the language associated with the target and each of the neighboring cognates. This could be simply the language name itself, but it is better to encode the name as a unique identifier (e.g. a short sequence of arbitrary symbols, such as emoji), so that the model does not learn spurious associations between the name of the language and cognate forms.[2]

The model used here slightly differs from the version used for geographic names in that the language identifiers and cognate forms are interleaved and then concatenated together into a single one dimensional tensor, interleaved with ids for each cognate in the set. This allows the model to better attend to the individual cognate and to copy the cognate as needed.

For the transformer model, the provided data sets are far too small for the model to learn anything. We therefore augmented the data in two ways. First, since one cannot assume that in the test set one will find cognate forms for all neighbor languages for a given target language, we also augmented the data by randomly removing neighbors, thus making new neighborhoods for the same cognate set, which lacked one or more of the neighbor cognate examples. In our experiments we generated 500 such random subsets for each original neighborhood, of which about half were randomly copies of the original neighborhood.

Second, in addition to the actual provided cognate groups, we synthesized similar cognates for each of the "neighbor" languages and the target. Two methods were used, the first being a simple pair unigram model and the second a bigram model that we will briefly mention in Section 5. For the unigram model, a Levenshtein-distance alignment was computed between each pair of cognates for all language pairs, and correspondences between IPA symbols were counted. These unigram counts were then used to randomly generate new pairs of "cognates". We generated ten random neighborhoods of this kind for each real or subset neighborhood as described above. For example, for the English set in the LISTSAMPLESIZE

[2]For example, in the Northern Pakistan set, there are number of variants of Balti in the set, and we wish to avoid the model learning to depend on the string *Balti* in making its predictions. See Table 4 in Appendix A for an example.

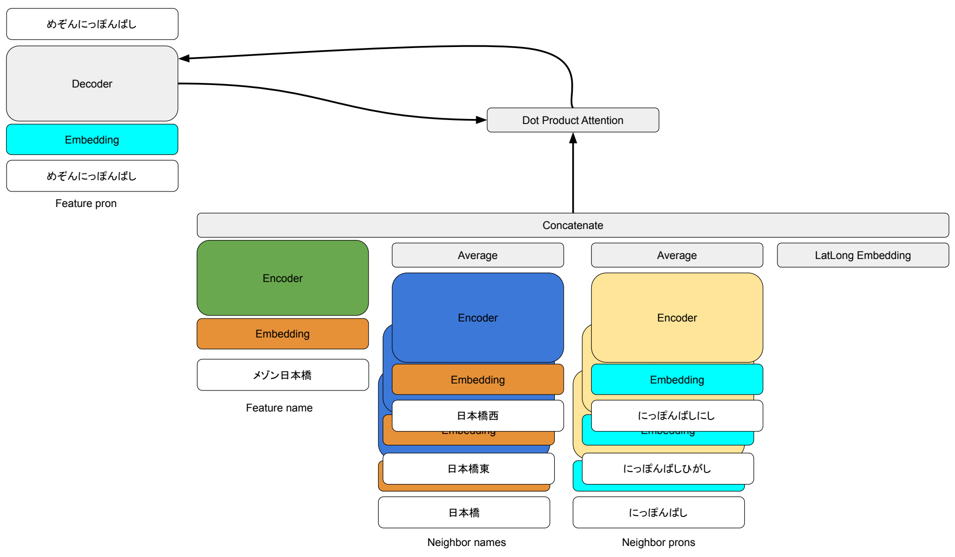


Figure 1: The transformer neighbors model, showing how the main feature and neighbor features are encoded. Colors for the embeddings reflect the shared embedding structure for the Transformer model. Example shown is メゾン日本橋 *mezon nipponbashi*, and some neighboring features 日本橋 *nipponbashi*, 日本橋西 *nipponbashi nishi* and 日本橋東 *nipponbashi higashi*.

data set, the original approximately 300 neighborhoods was expanded into about 4.2 million neighborhoods.

3.2 Model Details

The neighbors model is implemented using Lingvo (Shen et al., 2019), which is a framework for building neural networks in TensorFlow (Abadi et al., 2016), particularly sequence models. The core architecture of our transformer model derives from the Lingvo implementation of the neural machine translation system by Chen et al. (2018).[3]

We use the compact transformer configuration because the amount of data available for this task even after data augmentation is small. In our configuration, the encoders and decoders in Figure 1 use multi-head attention mechanism with two attention heads (Vaswani et al., 2017). There are three transformer layers, and the dimension of the feed-forward layer as well as the embedding dimension are set to 32. Dropout is applied during training with probability of 0.1 to residual layers, attention weights, and each feed-forward layer of the Transformer. Finally, we employ label smoothing to the decoder outputs, where the probability for correct class labels is reduced by the uncertainty factor $\epsilon = 0.2$ and for all other cases increased by ϵ/K, where K is the size of the vocabulary (Szegedy et al., 2016).

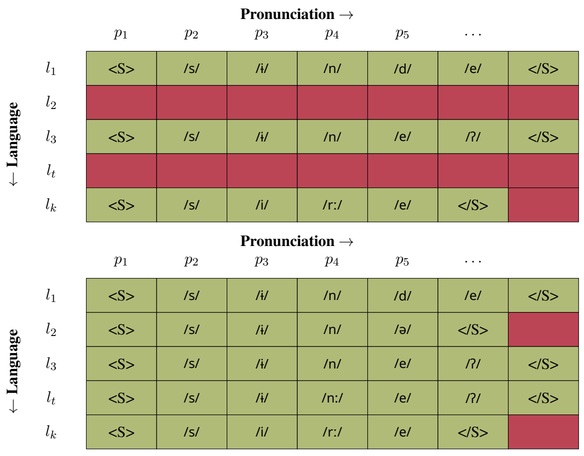
For training, we optimize sparse categorical cross-entropy loss using Adam procedure with initial learning rate $\alpha = 0.001$ and the parameters $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$ (Kingma and Ba, 2015). We optimize the word error error (WER) between the sequences of ground truth and the predicted phonemes. The batch size is set to 32 examples. No development set was set aside from the training data because we did not perform any parameter tuning. The latest and not necessarily best (according to the test set) checkpoints were chosen after the training ran for a specified number of steps. We employ beam search with the beam width of 8 during inference.

3.3 Open-Source Implementation Notes

The code publicly released for the shared task is mostly identical to our internal version used to produce the shared task results, but with some notable differences. First, in the open-source verson we do not encode the language names using unique emojis. Second, we used the development versions of TensorFlow and Lingvo integrated with XManager,[4] a platform for managing distributed machine learning experiments, for our internal experiments. This implies that our main results may

[3]https://github.com/tensorflow/lingvo/tree/master/lingvo/tasks/mt

[4]https://github.com/deepmind/xmanager



Pronunciation →

↓ Language

	p_1	p_2	p_3	p_4	p_5	$\cdots$	
l_1	<S>	/s/	/ɨ/	/n/	/d/	/e/	</S>
l_2	🟥	🟥	🟥	🟥	🟥	🟥	🟥
l_3	<S>	/s/	/ɨ/	/n/	/e/	/ʔ/	</S>
l_t	🟥	🟥	🟥	🟥	🟥	🟥	🟥
l_k	<S>	/s/	/i/	/rː/	/e/	</S>	🟥

Pronunciation →

↓ Language

	p_1	p_2	p_3	p_4	p_5	$\cdots$	
l_1	<S>	/s/	/ɨ/	/n/	/d/	/e/	</S>
l_2	<S>	/s/	/ɨ/	/n/	/ə/	</S>	🟥
l_3	<S>	/s/	/ɨ/	/n/	/e/	/ʔ/	</S>
l_t	<S>	/s/	/ɨ/	/nː/	/e/	/ʔ/	</S>
l_k	<S>	/s/	/i/	/rː/	/e/	</S>	🟥

Figure 2: (Top) Cognate set input represented as an "image". The individual pixel coordinates (x, y) correspond to (l, p), where l_i is a language and p_j is a phoneme. The pronunciations in this example are taken from FELEKESEMITIC cognate set for cognate ID 638. Short cognates are marked with padding. One or more cognates may be masked out entirely. (Bottom) Equivalent output image. Missing cognate "pixels" have been restored.

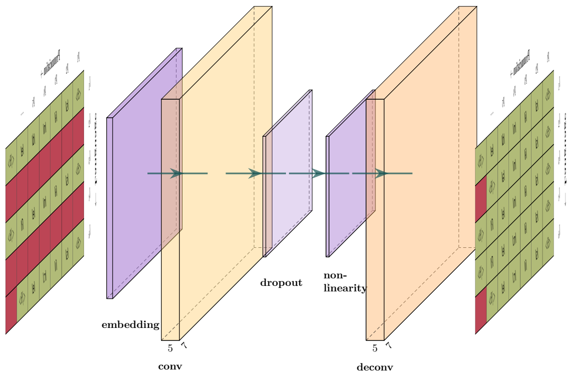


Figure 3: Simplified inpainting CNN architecture.

not be exactly reproducible with the open-source system, something that we discuss further in Section 5.

4 Cognate Reconstruction as Image Inpainting

The neighborhood model presented above casts the reflex generation task as a straightforward extension of Seq2Seq modeling. Here, we take a different tack, noting that if we treat the entire set of cognates as a unit (an "image"), then the task of generating unknown cognates is analogous to recovering missing/masked/corrupted parts of that image. In the field of image reconstruction, this is sometimes referred to as image inpainting (Qin et al., 2021; Jam et al., 2021; Peng et al., 2021). One of the popular state-of-the art methods employs convolutional neural networks (CNNs) to recover missing pixel values. In this work we apply one such architecture by Liu et al. (2018) from NVIDIA to cognate generation. As we see below, cognate set "images" contain relatively few "pixels", so we can get away with small networks with a single pair or convolution and deconvolution layers that are fast to train and evaluate, even on CPU.

4.1 Model Details

The model's input and output structure are shown in Figure 2. Input cognates are book-ended with start and end symbols, padded to a fixed length (20 in all our experiments), and stacked to form a grid. Crucially some cognates may be masked out when forming an input image, resulting in rows of nothing but padding. Each symbol is embedded, resulting in a data structure with n embedding dimensions per symbol, corresponding to the "channels" in an image[5]. Optionally, this image may be scaled by a constant factor equivalent to the total number of languages divided by the number of languages present. This ensures constant total "brightness" no matter how many cognates are masked out.

The image is then processed by a 2D convolution layer, with kernel height fixed to the number of languages in the cognate set, and kernel width determined by a hyperparameter. Convolution is followed by dropout and a nonlinearity, after which a deconvolution layer recovers logits at each pixel position for the available character set. The logits, in turn, can be used to predict the most likely character at each position, or to calculate a sparse categorical cross-entropy loss during training, given a target symbol. The simplified diagram of the model is shown in Figure 3.

The convolution operation for a given kernel with weights W is shown below, and mirrors that used in NVIDIA's paper (Liu et al., 2018). Here X is the set of input pixels, which are multiplied pointwise with a binary mask M, where missing cognate positions are set to 0. The result is scaled

[5]If the input was a standard RGB image, these would be values for red, green, and blue color intensity. Here, embedding dimensions don't necessarily correspond to any real-world scale.

by a factor equivalent to the sum of what the mask would look like if all languages were present (all values set to 1) divided by the sum of actual binary mask:

$$x' = \begin{cases} W^T(X \odot M)\frac{\text{sum}(1)}{\text{sum}(M)}, & \text{if sum}(M) \geq 1 \\ 0, & \text{otherwise} \end{cases}$$

4.2 Training Regime

As one goal of the image inpainting approach was to keep it as simple as possible, no synthetic data augmentation was used during training — just the base training data made available for the SIGTYP shared task.

For each language family, the available training data was further broken down by a random 80%/20% split into a base cognate training set, and a development set used for tuning. For the development set, each group of cognates was broken down into multiple dev samples by setting each available cognate as the reconstruction target in turn (replacing it with '?'). As the overall datasets available are small, each dev set held out in this way will only cover a few cognate groups, introducing significant model bias when used for parameter tuning. To counteract this, we generate ten different train/dev splits at random, which were used for ensembling as described below.

Given a train/dev split, actual training proceeded as follows. For S training steps per epoch, a random cognate group was drawn from the available training set, and a random subset of the cognates present was masked out (at least one cognate always remained present so the model would have some information to work with). This masked sample was fed as an image to the inpainting model, whose task was to recover the entire cognate group. For backpropagation, cross-entropy loss was calculated only for rows of the image where a cognate was present in the original cognate group — all other positions contributed zero loss since there was no target information available.

For each training step, parameter updates were performed using the Adam procedure (Kingma and Ba, 2015) with default TensorFlow parameters. Training was run for a total of 150 epochs consisting of 500 steps each. After each epoch, the exact-match word error rate (WER) was calculated for each sample in the dev set for each language, and a macro average across of the per-language WER values was taken. Checkpoints were saved only in cases where macro-WER performance on the dev set improved.

Name	Type	Alias
Mockingbird-I1	Inpainting	I1
Mockingbird-N1-A	Neighbors	N1-A
Mockingbird-N1-B	Neighbors	N1-B
Mockingbird-N1-C	Neighbors	N1-C
Mockingbird-N2	Neighbors	N2

Table 1: Five system configurations submitted to the shared task. The third column contains a shorthand of the full configuration name that we use in this paper.

For each train/dev split, a separate model was prepared with its own set of tuned hyperparameters. Hyperparameter tuning was done using Google's Vizier smart grid search procedure (Golovin et al., 2017),[6] which optimized macro-WER on the dev set for 100 total Vizier trials, with at most 10 trials running in parallel at any one time. The list of tunable model hyperparameters is provided in Appendix B.

4.3 Open-Source Implementation Notes

The core CNN model implementation in Tensorflow (Abadi et al., 2016) has been released as is and can be used for training and inference. We are not releasing the scaffolding required for integration with Vizier hyperparameter tuning because our internal implementation is quite different from the publicly available version. However, the tuned set of hyperparameters (residing in `hparams.json`) for each model have been released together with the results. These hyperparameters can be used to train models that should perform similarly to those described here.[7] It is also possible to implement an alternative smart grid search procedure using Keras Tuner (Shawki et al., 2021)[8] or any other framework for hyper-parameter optimization and neural architecture search that supports Tensorflow (Menghani, 2021).

5 Results and Discussion

The SIGTYP 2022 Shared Task was evaluated over 20 language-family specific datasets taken from the LexiBank repository (List et al., 2021).

[6] https://cloud.google.com/ai-platform/optimizer/docs/overview

[7] Trained model checkpoints are also available upon request — they were not included in the results package due to file size considerations.

[8] https://keras.io/keras_tuner/

Each dataset consisted of a series of cognate groups presented in CLDF (Forkel et al., 2018) format, with each cognate form stored as an IPA phonetic transcription. 10 datasets were provided for system development, and 10 "surprise" language families were held aside for final evaluation. Furthermore, each dataset was provided in five sparsity conditions (dropping 10%, 20%, 30%, 40%, and 50% of the available cognate forms). For brevity, we only discuss the 10% ("dense") and 50% ("sparse") conditions in this paper.[9]

We submitted five system configurations, shown in Table 1, to the shared task.[10] The first configuration `i1` is the inpainting CNN approach described in Section 4, while the rest of configurations are the Neighbors Transformer models introduced in Section 3.

There are four Neighbors Transformer configurations. The first three configurations (`N1-A`, `N1-B` and `N1-C`) correspond to the models built using our internal pipeline. The only difference between these configurations is the number of training steps: 25,000 (`N1-A`), 35,000 (`N1-B`), and 100,000 (`N1-C`). For these configurations we mapped the language names to unique emojis during training and inference.

The final Transformer configuration `N2` corresponds to the model trained using the released open-source pipeline. The training regime has some notable differences with the other Transformer configurations. First, as noted above, no language name-to-emoji mapping was performed. Second, when generating each random example $\{(p_t, l_t), (p_n, l_n)\}$ consisting of target t and neighbor n pronunciation/language pairs, the neighbor pronunciation p_n is randomly generated using a first-order Markov chain trained from all the bigrams constructed from the pronunciations available for language l_n, as opposed to unigrams used by the `N1` systems. Also, the target pronunciation p_t is generated from p_n by sampling from the distribution obtained using sound class-based pairwise alignment algorithm implementation from (List and Forkel, 2021) described in (List et al., 2018). Finally, the stopping criterion for the training process was rather ad-hoc: training for each language group was stopped after eyeballing the training set loss. Unlike the rest of the submitted systems, the `N2` configuration was only trained in the 10% ("dense") sparsity condition.

For the inpainting model, results were produced via a majority ensemble. For each language family, ten models were trained using the procedure described above corresponding to ten random 80%/20% train/dev splits of the available training data. Predictions for each test sample were obtained from each of the models, and the most common prediction across the set was retained.

Model results were evaluated according to four metrics selected by the shared task organizers. The first two are raw and normalized (divided by the number of characters in the target form) Levenshtein edit distances (Levenshtein, 1966) between the predicted and expected test cognate forms. For these metrics, lower is better as it indicates a closer match. The third metric, B-Cubed F-Scores, is designed to avoid overly penalizing systematic errors a system might make, discounting errors that occur across multiple trials (List, 2019b). The metric derives from the B-Cubed measure (Amigó et al., 2009) frequently used in historical linguistics to evaluate automatic cognate detection techniques (Hauer and Kondrak, 2011). For this metric, higher is better. Finally, standard BLEU scores (Papineni et al., 2002) as used when evaluating machine translation (again, higher is better) were included.

Tables 2 and 3 summarize the evaluation in the dense and sparse data conditions. Overall, both the Inpainting and Neighbor `N1` models match or improve upon the baseline method provided by the Shared Task organizers. The Inpainting model shows an overall advantage – its simplicity might mitigate against overfitting in such small datasets, but this isn't universal across all language families, and wanes as the task becomes more difficult moving from the dense to the sparse condition. In particular, the Neighbor models are very effective in the FELEKESEMITIC language family in the sparse data condition, while the Inpainting model behaves at baseline level. This could be due to the unique morphology of Semitic languages, and the ability of the N1 models to use contextual cues that the simpler model architectures aren't able to represent, but which compensate for data sparsity to an extent. The condition may also benefit more than most from the data augmentation strategy used in training the Neighbor models.

In terms of overall results on the "surprise"

[9] Please see `https://github.com/sigtyp/ST2022/tree/main/results` for all the available results.

[10] `https://github.com/sigtyp/ST2022/tree/main/systems`

	BL	I1	N1-A	N1-B	N1-C	N2
dev total	1.34 0.28 0.72 0.61	1.05 0.23 0.74 0.68	1.25 0.27 0.72 0.63	1.31 0.28 0.70 0.61	1.29 0.28 0.69 0.62	1.37 0.29 0.68 0.60
davletshinaztecan	2.07 0.33 0.64 0.52	1.87 0.30 0.66 0.56	2.04 0.33 0.63 0.53	2.20 0.36 0.62 0.49	1.94 0.32 0.64 0.56	2.28 0.38 0.59 0.45
felekesemitic	1.46 0.27 0.69 0.59	1.29 0.24 0.72 0.65	1.68 0.31 0.64 0.55	1.76 0.33 0.63 0.52	1.92 0.36 0.60 0.48	1.88 0.36 0.59 0.49
hantganbangime	1.31 0.33 0.62 0.54	1.12 0.29 0.64 0.58	1.28 0.33 0.61 0.54	1.32 0.34 0.60 0.53	1.47 0.37 0.56 0.49	1.57 0.38 0.54 0.47
hattorijaponic	0.91 0.19 0.80 0.73	0.71 0.16 0.83 0.78	0.94 0.20 0.80 0.72	0.92 0.20 0.78 0.74	0.88 0.19 0.79 0.75	1.12 0.21 0.75 0.72
listsamplesize	3.34 0.62 0.41 0.22	2.35 0.46 0.50 0.40	2.80 0.54 0.50 0.33	2.79 0.55 0.49 0.32	2.54 0.52 0.49 0.34	2.59 0.50 0.48 0.37
backstromnorthernpakistan	0.89 0.18 0.86 0.72	0.60 0.12 0.87 0.80	0.83 0.17 0.82 0.72	0.83 0.18 0.83 0.72	0.81 0.17 0.82 0.73	0.79 0.16 0.83 0.76
mannburmish	1.98 0.52 0.51 0.32	1.55 0.42 0.57 0.43	1.74 0.47 0.53 0.39	1.89 0.50 0.52 0.36	1.93 0.50 0.52 0.35	1.95 0.52 0.51 0.31
castrosui	0.16 0.04 0.95 0.94	0.14 0.03 0.95 0.95	0.16 0.04 0.95 0.94	0.15 0.03 0.95 0.95	0.20 0.05 0.93 0.92	0.29 0.07 0.91 0.89
allenbai	0.72 0.23 0.77 0.68	0.55 0.18 0.80 0.75	0.58 0.19 0.79 0.74	0.64 0.21 0.78 0.71	0.70 0.23 0.73 0.69	0.67 0.22 0.77 0.70
abrahammonpa	0.55 0.12 0.90 0.81	0.34 0.06 0.91 0.90	0.47 0.09 0.87 0.86	0.55 0.11 0.84 0.83	0.51 0.09 0.86 0.85	0.53 0.10 0.86 0.84
surprise total	1.21 0.31 0.72 0.57	0.92 0.24 0.77 0.66	1.02 0.26 0.76 0.65	1.04 0.26 0.76 0.64	1.13 0.29 0.73 0.61	1.21 0.31 0.71 0.57
beidazihui	1.10 0.30 0.73 0.58	0.50 0.14 0.84 0.80	0.48 0.13 0.86 0.81	0.40 0.11 0.87 0.84	0.45 0.12 0.86 0.83	1.13 0.29 0.70 0.60
hillburmish	1.18 0.32 0.66 0.57	1.06 0.29 0.68 0.61	1.13 0.30 0.66 0.60	1.13 0.30 0.66 0.60	1.37 0.37 0.62 0.53	1.50 0.39 0.59 0.48
bodtkhobwa	0.49 0.20 0.76 0.72	0.39 0.16 0.80 0.78	0.25 0.11 0.88 0.85	0.26 0.11 0.87 0.85	0.42 0.18 0.77 0.77	0.68 0.28 0.67 0.62
bantubvd	1.12 0.26 0.79 0.62	0.89 0.22 0.80 0.68	1.01 0.25 0.82 0.63	1.03 0.26 0.82 0.62	0.98 0.25 0.81 0.63	1.10 0.27 0.77 0.60
bremerberta	1.72 0.32 0.71 0.51	1.16 0.21 0.77 0.66	1.35 0.25 0.74 0.61	1.35 0.25 0.74 0.61	1.47 0.27 0.71 0.58	1.26 0.23 0.75 0.66
deepadungpalaung	1.07 0.42 0.76 0.44	0.55 0.22 0.89 0.70	0.73 0.27 0.85 0.63	0.88 0.32 0.82 0.57	0.92 0.34 0.80 0.54	0.86 0.31 0.83 0.58
luangthongkumkaren	0.38 0.10 0.91 0.84	0.36 0.09 0.90 0.86	0.26 0.07 0.92 0.89	0.29 0.08 0.91 0.88	0.33 0.09 0.89 0.87	0.35 0.10 0.90 0.85
birchallchapacuran	1.63 0.31 0.65 0.54	1.57 0.30 0.65 0.56	2.04 0.37 0.57 0.47	2.01 0.36 0.58 0.48	2.01 0.37 0.58 0.48	2.01 0.39 0.57 0.45
wangbai	0.62 0.18 0.80 0.73	0.49 0.14 0.83 0.79	0.48 0.14 0.83 0.80	0.53 0.16 0.81 0.77	0.61 0.18 0.78 0.74	0.62 0.18 0.80 0.74
kesslersignificance	2.77 0.70 0.47 0.17	2.23 0.67 0.51 0.20	2.49 0.67 0.49 0.18	2.55 0.68 0.50 0.18	2.69 0.69 0.47 0.17	2.60 0.71 0.47 0.16

Table 2: Results by model for the 0.10 data condition (BL=Baseline, I1=Inpainting, N*=Neighborhood model), averaged by language group. The four values per entry cover the four metrics used in the shared task (black=edit distance, olive=normalized edit distance, red=B-Cubed F-Score, blue=BLEU).

	BL	I1	N1-A	N1-B	N1-C
dev total	1.75 0.37 0.60 0.50	1.40 0.31 0.63 0.58	1.60 0.34 0.59 0.54	1.61 0.34 0.58 0.53	1.63 0.35 0.57 0.52
davletshinaztecan	2.09 0.36 0.59 0.48	1.69 0.30 0.63 0.56	2.29 0.38 0.54 0.46	2.44 0.40 0.51 0.44	2.21 0.37 0.54 0.48
felekesemitic	2.90 0.53 0.45 0.28	2.85 0.53 0.41 0.31	2.33 0.41 0.49 0.42	2.27 0.41 0.49 0.42	2.19 0.40 0.50 0.44
hantganbangime	1.98 0.48 0.44 0.36	1.38 0.36 0.53 0.50	1.65 0.42 0.48 0.43	1.65 0.42 0.47 0.43	1.86 0.47 0.44 0.37
hattorijaponic	1.50 0.30 0.65 0.58	1.33 0.27 0.69 0.63	1.66 0.32 0.60 0.57	1.57 0.31 0.61 0.59	1.50 0.30 0.63 0.60
listsamplesize	3.68 0.69 0.37 0.17	2.43 0.51 0.42 0.35	2.72 0.53 0.41 0.31	2.71 0.54 0.41 0.30	2.81 0.54 0.40 0.30
backstromnorthernpakistan	0.97 0.22 0.77 0.66	0.73 0.17 0.79 0.74	1.00 0.21 0.72 0.67	1.03 0.22 0.71 0.65	0.93 0.21 0.73 0.68
mannburmish	2.33 0.60 0.36 0.25	2.02 0.55 0.39 0.30	2.41 0.62 0.34 0.24	2.32 0.62 0.34 0.25	2.38 0.62 0.33 0.25
castrosui	0.39 0.10 0.88 0.84	0.29 0.07 0.90 0.88	0.32 0.08 0.89 0.87	0.34 0.08 0.88 0.87	0.41 0.10 0.85 0.84
allenbai	0.76 0.25 0.71 0.66	0.64 0.21 0.75 0.71	0.78 0.25 0.68 0.65	0.84 0.27 0.66 0.64	0.99 0.32 0.59 0.57
abrahammonpa	0.94 0.18 0.76 0.72	0.66 0.12 0.80 0.80	0.87 0.16 0.74 0.74	0.95 0.18 0.72 0.72	1.03 0.20 0.70 0.70
surprise total	1.89 0.44 0.56 0.43	1.42 0.35 0.61 0.53	1.55 0.38 0.60 0.49	1.51 0.37 0.59 0.50	1.58 0.40 0.56 0.47
bremerberta	2.49 0.46 0.53 0.35	1.58 0.30 0.62 0.56	1.99 0.38 0.56 0.46	1.85 0.35 0.58 0.49	2.05 0.39 0.55 0.44
wangbai	1.02 0.29 0.66 0.58	0.97 0.28 0.66 0.60	1.05 0.31 0.63 0.58	1.06 0.31 0.62 0.57	1.15 0.34 0.59 0.54
luangthongkumkaren	0.66 0.17 0.81 0.73	0.55 0.15 0.80 0.78	0.56 0.15 0.80 0.77	0.62 0.17 0.78 0.75	0.77 0.21 0.73 0.69
hillburmish	2.10 0.54 0.47 0.34	1.64 0.44 0.51 0.45	2.66 0.68 0.44 0.17	1.87 0.49 0.48 0.38	1.80 0.47 0.46 0.39
birchallchapacuran	3.17 0.47 0.48 0.32	2.81 0.47 0.44 0.35	2.80 0.47 0.44 0.34	2.80 0.47 0.45 0.33	2.83 0.48 0.44 0.33
bantubvd	1.99 0.45 0.56 0.39	1.53 0.36 0.61 0.50	1.29 0.31 0.69 0.55	1.43 0.34 0.65 0.51	1.45 0.35 0.64 0.50
kesslersignificance	4.06 0.89 0.29 0.04	2.85 0.73 0.30 0.15	2.77 0.67 0.34 0.17	2.81 0.68 0.34 0.16	2.95 0.71 0.33 0.14
bodtkhobwa	0.63 0.27 0.66 0.65	0.53 0.22 0.70 0.71	0.56 0.23 0.68 0.69	0.66 0.27 0.63 0.64	0.77 0.32 0.57 0.59
beidazihui	1.15 0.32 0.67 0.56	0.48 0.13 0.80 0.80	0.45 0.13 0.82 0.82	0.48 0.13 0.80 0.81	0.57 0.16 0.77 0.77
deepadungpalaung	1.63 0.58 0.50 0.30	1.23 0.44 0.60 0.43	1.39 0.48 0.57 0.39	1.49 0.52 0.54 0.36	1.47 0.52 0.53 0.35

Table 3: Results by model for the 0.50 data condition (BL=Baseline, I1=Inpainting, N*=Neighborhood model), averaged by language group. The four values per entry cover the four metrics used in the shared task (black=edit distance, olive=normalized edit distance, red=B-Cubed F-Scores, blue=BLEU).

sets, the worst performing configuration is the N2 model. It performs significantly worse than the N1 Neighbor and the Inpainting configurations using the edit distance-based metrics and decidely worse than the Inpainting method using B-Cubed F-Scores and BLEU. We hypothesize the existence of two confounding factors that may be affecting the model's performance. First, we trained it using significantly smaller (compared to N1 systems) amounts of augmented data. In addition, stopping the training process after a random number of steps may have resulted in under-training. Analysis of N2 model's results on individual language groups displays the uneven performance of this model. On the BremerBerta and DeepadungPalaung sets, the model strongly outperforms the baseline and improves upon one of the N1 configurations, while at the same time being significantly worse than the baseline on the HillBurmish set.

6 Conclusions

We presented two approaches to the problem of cognate reflex prediction, one based on the transformer Seq2Seq architecture, and one based on convolutional networks. Both approaches stem from natural, intuitive interpretations of the problem. The "neighbors" transformer approach treats the problem as one of reconstructing the phonetic form of a cognate by considering all the other cognates in the set, on analogy to the problem of reconstructing the reading of a geographical feature on the basis of the pronunciation of names of geographic neighbors. The inpainting approach treats the problem as being similar to filling in missing pixels in an image on the basis of the surrounding context pixels. We submitted 5 system variants (1 convolutional model and 4 transformer models) to the SIGTYP 2022 Shared Task, where they performed well relative to the provided baseline and other submissions.

Supplementary Material

The implementation of the Inpainting CNN and the Neighbors transformer models described in this work is available in Google Research repository on GitHub (https://github.com/google-research/google-research/tree/59f02a3cb447f381a7450c89f37dda042819216e/cognate_inpaint_neighbors). The results for these systems are curated on GitHub (https://github.com/sigtyp/ST2022) along with the results of the other systems submitted to the shared task, and have been archived with Zenodo (https://doi.org/10.5281/zenodo.6538626).

Acknowledgements

We would like to thank Llion Jones who developed the Transformer version of the neighbors model for the geographic names task described in Section 3.1. We also thank Brian Roark for useful feedback on an earlier version of this paper.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, Xiaoqiang Zheng, et al. 2016. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, Savannah, GA, USA.

Enrique Amigó, Julio Gonzalo, Javier Artiles, and Felisa Verdejo. 2009. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information retrieval*, 12(4):461–486.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2018. The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–86, Melbourne, Australia. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Michael Covington. 1996. An algorithm to align words for historical comparison. *Computational Linguistics*, 22(4):481–496.

Robert Forkel, Johann-Mattis List, Simon J Greenhill, Christoph Rzymski, Sebastian Bank, Michael Cysouw, Harald Hammarström, Martin Haspelmath, Gereon A Kaiping, and Russell D Gray. 2018. Cross-linguistic data formats, advancing data sharing and re-use in comparative linguistics. *Scientific data*, 5(1):1–10.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and David Sculley. 2017. Google Vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*, pages 1487–1495, Halifax, NS, Canada.

Bradley Hauer and Grzegorz Kondrak. 2011. Clustering semantically equivalent words into cognate sets in multilingual lists. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 865–873, Chiang Mai, Thailand. Asian Federation of Natural Language Processing.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Jireh Jam, Connah Kendrick, Kevin Walker, Vincent Drouard, Jison Gee-Sern Hsu, and Moi Hoon Yap. 2021. A comprehensive review of past and present image inpainting methods. *Computer Vision and Image Understanding*, 203:103147.

Llion Jones, Richard Sproat, and Haruko Ishikawa. 2022. Helpful neighbors: Leveraging geographic neighbors to aid in placename pronunciation. In preparation.

William Jones. 1786. Third anniversary discourse to the Asiatic Society, Calcutta.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR)*, San Diego, CA, USA.

Grzegorz Kondrak. 2000. A new algorithm for the alignment of phonetic sequences. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*, pages 288–295, San Francisco, CA. ACL, Morgan Kaufmann.

Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics — Doklady*, 10(8):707–710.

Johann-Mattis List. 2019a. Automatic inference of sound correspondence patterns across multiple languages. *Computational Linguistics*, 45(1):137–161.

Johann-Mattis List. 2019b. Beyond edit distances: Comparing linguistic reconstruction systems. *Theoretical Linguistics*, 45(3-4):247–258.

Johann-Mattis List and Robert Forkel. 2021. LingPy. A Python library for historical linguistics. July, version 2.6.8, `https://github.com/lingpy/lingpy`.

Johann-Mattis List, Robert Forkel, Simon J Greenhill, Christoph Rzymski, Johannes Englisch, and Russell D Gray. 2021. Lexibank: A public repository of standardized wordlists with computed phonological and lexical features. *Scientific Data*. To appear.

Johann-Mattis List, Mary Walworth, Simon J. Greenhill, Tiago Tresoldi, and Robert Forkel. 2018. Sequence comparison in computational historical linguistics. *Journal of Language Evolution*, 3(2):130–144.

Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. 2018. Image inpainting for irregular holes using partial convolutions. In *Proceedings of the 15th European Conference on Computer Vision (ECCV 2018)*, pages 89–105, Munich, Germany. Springer International Publishing. Preprint.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, volume 28, Atlanta, Georgia, USA.

Carlo Meloni, Shauli Ravfogel, and Yoav Goldberg. 2021. Ab antiquo: Neural proto-language reconstruction. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4460–4473, Online. Association for Computational Linguistics.

Gaurav Menghani. 2021. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *arXiv preprint arXiv:2106.08962*.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 807–814, Haifa, Israel. Association for Computing Machinery (ACM).

Keiron O'Shea and Ryan Nash. 2015. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Hermann Paul. 1880. *Prinzipien der Sprachgeschichte*. Max Niemeyer, Halle.

Jialun Peng, Dong Liu, Songcen Xu, and Houqiang Li. 2021. Generating diverse structure for image inpainting with hierarchical VQ-VAE. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10775–10784, Nashville, TN, USA. IEEE.

Zhen Qin, Qingliang Zeng, Yixin Zong, and Fan Xu. 2021. Image inpainting based on deep learning: A review. *Displays*, 69:102028.

N. Shawki, R. Rodriguez Nunez, I. Obeid, and J. Picone. 2021. On automating hyperparameter optimization for deep learning applications. In *Proceedings of 2021 IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*, pages 1–7, Philadelphia, PA, USA. IEEE.

Jonathan Shen, Patrick Nguyen, Yonghui Wu, Zhifeng Chen, Mia X. Chen, Ye Jia, Anjuli Kannan, Tara Sainath, Yuan Cao, Chung-Cheng Chiu, Yanzhang He, Jan Chorowski, Smit Hinsu, Stella Laurenzo, James Qin, Orhan Firat, Wolfgang Macherey, Suyog Gupta, Ankur Bapna, Shuyuan Zhang, Ruoming Pang, Ron J. Weiss, Rohit Prabhavalkar, Qiao Liang, Benoit Jacob, Bowen Liang, HyoukJoong Lee, Ciprian Chelba, et al. 2019. LINGVO: A modular and scalable framework for sequence-to-sequence modeling. *arXiv preprint arXiv:1902.08295*.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS'14)*, pages 3104–3112. MIT Press.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception architecture for computer vision. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, Las Vegas, USA. IEEE.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017)*, pages 5998–6008, Long Beach, CA. Curran Associates Inc.

Language Name	Emoji
`ChorbatBalti`	👻
`KhapaluBalti`	👂
`KharmangBalti`	👃
`RonduBalti`	🧠
`ShigarBalti`	👅
`SkarduBalti`	👄
`SkarduPurki`	👶

Table 4: The one-to-one mapping between the names of languages in BACKSTROMNORTHERNPAKISTAN language group and the corresponding Unicode emojis.

A Language Name Emoji Mapping

The mapping between names of the Northern Pakistan languages as encoded in the shared task data for the BACKSTROMNORTHERNPAKISTAN language group and the emojis is shown in Table 4. The mapping takes place during the generation of the training data. The inverse mapping is applied during decoding at the inference stage to map the emojis back to language names.

B Tuning the Inpainting Model

For the cognate inpainting model there are six tunable hyperparameters:

- The symbol embedding dimension.
- The width w of the 2D convolution kernel (h, w), where h is the number of languages and w corresponds window of characters processed processed by each convolution and deconvolution operation.
- The number of convolution filters for the 2D convolution layer.
- Probability for the dropout layer that follows the convolution layer (Hinton et al., 2012).
- The nonlinearity activation applied to the convolved inputs after the dropout. This choice is between rectified linear units (ReLU) (Nair and Hinton, 2010), Leaky ReLU (Maas et al., 2013) and the hyperbolic tangent function.
- Whether to scale the embeddings, the outputs of the convolution layer or not to apply the scaling at all.