University Polytechnic of Valencia
Department of Computer Systems and Computation

**MASTER THESIS**


UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Jesús Vieco Pérez

# Obtaining n best alternatives for classifying Unicode symbols

Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

Tutors:
Joan Andreu Sánchez Peiró
Jose Miguel Benedí Ruiz

Valencia 2017

**Abstract:**
The Unicode character set has been increased in last years until grouping more than 100000 characters. We developed a classifier which can predict the n most probable solutions to a given handwritten character in a smaller Unicode set. Even with the size reduction we still have a classification problem with a big number of classes (5488 in total) without any training sample. Before dealing with this problem we performed some experiments on the UJI PEN dataset. In these experiments we used two different data generation techniques, distortions and variational autoencoders as generative models. We tried feature extraction methods with both offline and online data. The generation along with the feature extraction was tested in several models of neural networks like convolutional networks or LSTM.

**Keywords:** Unicode, character recognition, neural network, data generation.

**Resumen:**
El conjunto de caracteres Unicode se ha incrementado en los últimos años hasta llegar a agrupar más de 100000 caracteres. Hemos desarrollado un clasificador que puede predecir las n clases más probables de un carácter escrito a mano perteneciente a un conjunto más pequeño de caracteres Unicode. Incluso con la reducción de tamaño todavía tenemos un problema de clasificación con muchas clases (5488 en total) sin ninguna muestra de entrenamiento. Antes de tratar este problema hemos realizado algunos experimentos con el corpus UJI PEN. En estos experimentos hemos utilizado dos técnicas de generación de datos, distorsiones y el uso de *variational autoencoders* como modelos generativos. Hemos probado diferentes métodos de extracción de características tanto con datos *offline* como con datos *online*. La generación y la extracción de características han sido probadas en diferentes modelos de redes neuronales como las redes convolucionales o las LSTM.

**Palabras clave:** Unicode, reconocimiento de caracteres, redes neuronales, generación de datos.

# Contents

# List of Figures

# List of Tables

# 1. Introduction

At the end of the decade of 1980 Joseph D. Becker published a draft proposal for an international and multilingual encoding for text characters, which later was called Unicode. Some years later the Unicode Consortium was created like a non-profit organization that was in change of the development of the Unicode standard. In 1991 the Consortium published *The Unicode Standard* [3]. This publication contains 7,161 character symbols with its correspondent Unicode representation.

Nowadays Unicode is a computing industry standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems. The latest version of Unicode contains a repertoire of 128,237 characters and still increasing.

## 1.1 Problem description

Due to the large increase in the number of characters collected by the Unicode standard, searching for a character knowing only its graphic representation becomes a very expensive task. There is many web resources where all the possible Unicode characters are shown in a table. They make the task easy but not enough due to the large number of characters where we are looking for and the similarity between them.

This project belongs to a contract between the company Wiris [57] and the Universitat Politècnica de València (UPV). The goal is to develop a system which retrieves the n nearest characters to a one we drawn. To develop this system we need a classification system which receives online data and returns the n most probable classes. These classification system must be able to deal with a large amount of classes and be scalable due to the constant increasing of the amount of characters in the Unicode set. In addition the system must be capable of classify in a reasonable time to make the application useful.

In order to solve a simplified problem we reduced the whole Unicode list to a smaller character set. In this new list we have 5.488 characters listed in Annexed I: List of Unicode characters. This help us to reduce the dimensional problem but don't solve anything else. There is still so much classes and some of them represents the same character with a different font. In addition we don't have any set with training data which we can start working. So we need to resort to data generation techniques. We have to approximate the synthetic samples to real samples to simulate human writing to improve the classification.

## 1.2 Objectives

The main objectives of this work are the next four:

1. Obtain a set of labeled handwritten samples which contains all of our characters.

2. Find a good representation of this images to train an statistical model.

3. Find an statistical model capable of deal with many classes and capable of retrieve a score for every class to obtain the n with the higher score.

4. Find the best way to scale the previous model in the future to increase the number of classes where the model have to choose.

All of this objectives along with it associated problems will be discussed in the next sections in the same order as are proposed.

## 1.3 Related work

Character recognition systems have captured the attention of the AI research community in the last years. One of the most known character dataset in this scope is MNIST [30]. MNIST is a database of the digits between 0 and 9 which contains 70000 handwritten images. This dataset has been used and is still used a test set in scientific experiments. Whith this dataset several model have been tested to compare its performance. Some of this models are k-nearest neighbours , support vector machines and neural networks [32][27]. One of the best results is achieved with neural networks with a 0.21% of error [56] so we can consider that the task is solved but it is still used to try some different techniques.

MNIST dataset has not been used only to try classifier models, it was used to try different methods of synthetic data generation. One of this methods is the use of deformations like elastic distortions [46] or geometrical transformations [6]. Another is the use of generative models to generate new data training a model with the real one, we can find examples with variational autoencoders [13] or generative adversarial networks [14].

In our case we have two data sources where we can extract the features, online data and offline data. To extract online data features we can find several methods in literature, like the histograms of directions [33], the 8-directional features [7][5] or adding derivatives and curvature information for every point [4]. With the offline data some authors decided to use every pixel as a feature [29][27]. Some others used a complex method to extract features like scale invariant feature transform (SIFT) [59] or normalization-cooperated feature extraction (NCFE) [17].

# 2. Sample generation

One of the biggest problem in this classification task is the number of training samples that we have. We don't have any sample for training, instead of this we have every character rendered with a computer font style as we can see in the example of Figure 2.1. Due to the big number of classes making an acquisition of real samples become in a great expense of time or money. Instead of making an acquisition we will develop a generator of synthetic labeled samples which simulates the human handwriting starting from Unicode characters represented in a computer font style.

Figure 2.1: Unicode character example

In the literature we can find several ways to generate synthetic data, but we will focus on approaches based on character images due to the closeness to our classification problem. Some authors decide to generate new samples by means geometrical transformations applied to the original images, like translations, rotations, etc [52][26][53][54]. Even use these transformations along with some other operations like thinning/thickening [52][54] or elastic distortions [46]. Some others approaches are based on generative models like boltzmann machines [43], generative adversarial networks [14][36] or variational autoencoders [13]. We will try try two methods. The first method will use geometrical transformations applying elastic distortions before to the original image, we call this distortions method. The second method will use a variational autoencoder (VAE) 2.2 to generate new samples.

## 2.1 Distortions method

Distortions generation method is based on modifying an image using some deformations. We are going to list some of this deformations, the ones we used, and briefly describe how they are made. Then we show the process that we make to generate samples from an original sample to feed our statistical model. As we said before we are going to use the original image, then we will apply a elastic distortion on it and next we are going to use other deformations. This deformations are basically affine transformations and aspect ratio modifications. An affine transformation is a set of geometrical deformations. These deformations modify the geometrical structure of the image. The affine transformation technique is typically used to correct for geometric distortions or deformations that occur with non-ideal camera angles but in our case we will use it to do the opposite.

### 2.1.1 Elastic distortions

The elastic distortions [46] are local deformations which induces a "wiggly" effect in the image. The idea is generate two random displacement fields of size width and height of the image respectively, with random numbers. Once we have the displacement fields we convolved them with a Gaussian of standard deviation which must be manually tuned. With our new displacement fields we calculate the new values for every pixel in the image.

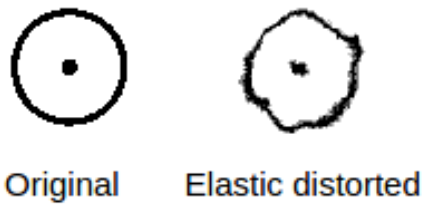


Figure 2.2: Elastic distortions on a character

We can see on Figure 2.2 the result of use the elastic distortions on a image. As we can see the deformation are nearest to a character which can be drawn by an human than the original. We are going to use the distorted image and apply other deformations to generate our base dataset to train the statistical model.

### 2.1.2 Rotation

Rotation is a circular transformation around a point or an axis. In our case, we will consider a rotation around an axis defined by the normal vector to the image plane located at the center of the image. Where the angle of rotation in our case will be between -30º and 30º. In the Figure 2.3 we can see an example of a rotation over an image.

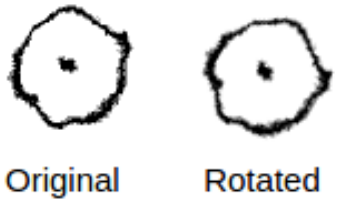


Figure 2.3: Rotation on a character

### 2.1.3 Shear

The transvection or shear mapping is another affine transformation which modify position of an object. Applying a shear map to a set of points of the plane will change all angles between them (except straight angles), and the length of any line segment that is not parallel to the direction of displacement. In our case we are going to share the same angle in both direction x and y to simulate an skew in

the image as we can see in Figure 2.4. We can shear an image to right or to left, in our case we use both to generate more samples and try to train our classifier in a more general way.

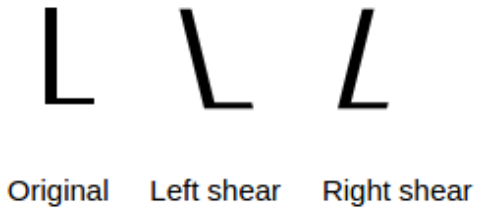


Figure 2.4: Shear on a character

### 2.1.4 Modifying aspect ratio

The aspect ratio (AR) is the relation between width and height of an image. The aspect ratio is expressed as two numbers separated by a colon. The values x and y do not represent actual widths and heights. In our case we are always have a square image were width is equal to height due to the feature extraction and the padding so our aspect ration will be always 1:1. Resizing our images changing the aspect ratio is another way to get new samples. We are going to generate samples as we can see in Figure 2.5, modifying the aspect ratio making the image higher or wider.

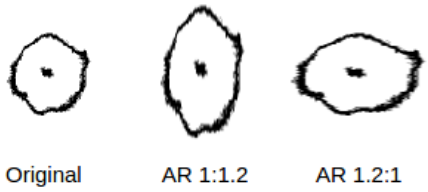


Figure 2.5: Aspect ratio modified

### 2.1.5 Generation pipeline

Several combinations of the previous distortions can be carried out. In our case we use a static generation and them apply a random rotation to every sample in order to generate different samples for every training step. This generation returns the following eighteen samples:

- The original sample
- The original sample with aspect ratio 1:1.2
- The original sample with aspect ratio 1.2:1
- The original sample with shear to the left
- The original sample with to the right

- The original sample with aspect ratio 1:1.2 and shear to the left
- The original sample with aspect ratio 1:1.2 and shear to the right
- The original sample with aspect ratio 1.2:1 and shear to the left
- The original sample with aspect ratio 1.2:1 and shear to the right
- The elastic distorted sample
- The elastic distorted sample with aspect ratio 1:1.2
- The elastic distorted sample with aspect ratio 1.2:1
- The elastic distorted sample with shear to the left
- The elastic distorted sample with to the right
- The elastic distorted sample with aspect ratio 1:1.2 and shear to the left
- The elastic distorted sample with aspect ratio 1:1.2 and shear to the right
- The elastic distorted sample with aspect ratio 1.2:1 and shear to the left
- The elastic distorted sample with aspect ratio 1.2:1 and shear to the right

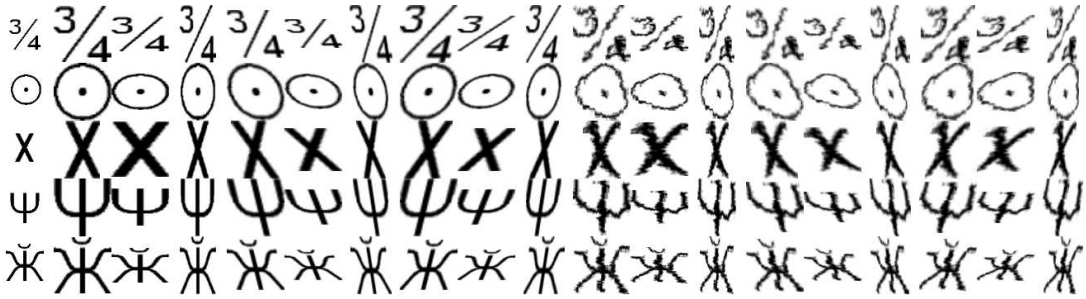

Figure 2.6: Example of generation using distortions

In Figure 2.6 we have a example of generation with this method. At the left we have the original image, then we got the eighteen images generated by our process explained before without including rotations. They are not ordered as in the list because we generate it in another order to reduce the memory usage.

## 2.2 Variational autoencoder

In this section instead of apply some transformations to our original images, we are going to use a generative model. This model will be a variational autoencoder (VAE) [25]. The VAE have two parts, encoder and decoder (Figure 2.7). The encoder have the task of take the raw image and compress it in a lower representation. The decoder take this representation and decompress it in the original image. The decoder and encoder stacks together and are training like a one single network using back-propagation [41][40] where the input and the output are the

original image. The point where encoder and decoder joins is the lower dimensional space and its called latent space. The key idea in variational autoencoders is that the latent space is forced to have a Gaussian distribution.

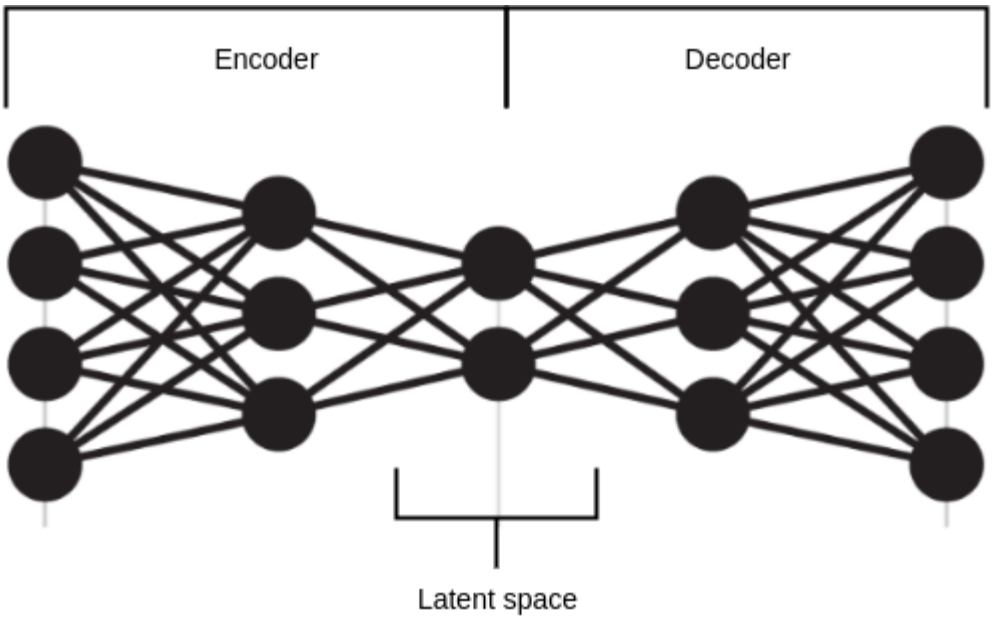


Figure 2.7: Variational autoencoder structure

To generate our samples once we have the VAE trained we take two original samples with the same label and compress it with our encoder. With the two representations in our latent space we make a linear interpolation between them. We take several points between both and decompress it obtaining new samples with the same label as the originals. In Figure 2.8 we have some examples of generation with this method. On the left and on the right we have the original samples and between them we have some samples obtained interpolating these originals in the latent space.

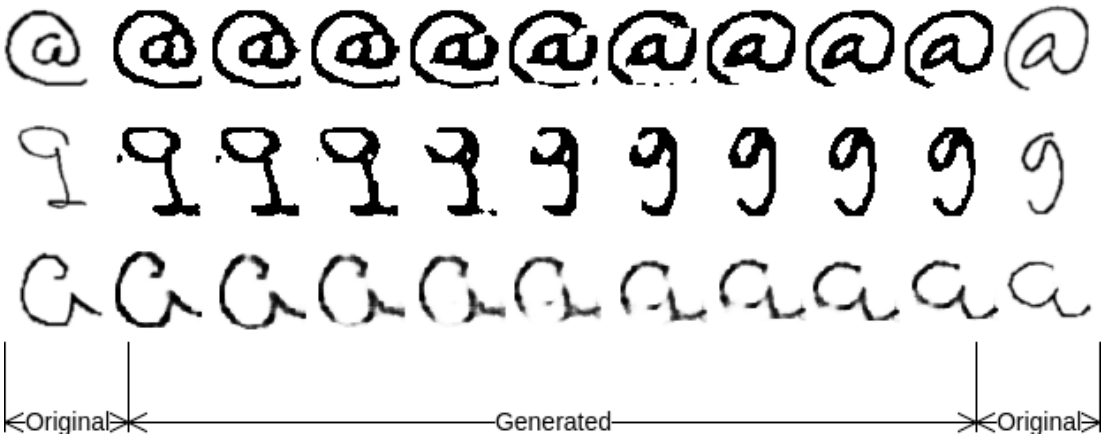


Figure 2.8: Example of generation with VAE

# 3. Object representation

In this chapter we are going to talk about how we preprocess the data to extract some features using in our statistical model. Feature extraction addresses the problem of finding the most compact and informative set of features, to improve the learning process. Defining feature vectors remains the most common and convenient means of data representation for classification problems. Choosing informative, discriminating and independent features is a crucial step for effective classification algorithms.

Our original data for every sample is a list of tuples representing the path followed to draw the character. Every tuple has got two elements, the first one is the position on the x-axis where the mouse is, the second element is the same but in the y-axis. If we draw a straight line between the points of every consecutive pair of tuples we got the image. As we are developing a classifier, we can use both, the raw image (offline data) and the list of tuples (online data). Due to the different preprocessing for each of them we are going to describe it separately.

## 3.1 Online data

The online data can be seen as the path point by point that we had followed to draw the full image. Every point have 2 values, its position of the x-coordinate or abscissa and its position of the y-coordinate or ordinate. The main problem of this representation is the variability of the length. That is, every sample has a different number of elements, this doesn't affect to our feature extraction method but it will do in our statistical model and we will see in the next chapter. If we look at the Figure 3.1 we can distinguish two different traces, one for the exterior circle and other for the letter C.

Figure 3.1: Copyright handwriting example

During the years many techniques have been proposed for the extraction of characteristics of these representation. One of the most used is based on the idea to codify the angle of the line which join two points and the x-axis in one of the 8-directional axes [5] [7]. Instead of this approach we tried another one using the first and the second derivatives [4]. So our online sample extraction system for samples will use:

- X-coordinate

- Y-coordinate
- New trace flag
- Normalized first derivative
- Normalized second derivative
- Curvature

Where the x-coordinate and the y-coordinate are normalized. The third parameter is a boolean value which indicates if a non consecutive line starts (the pen ups and go down in another point), these flag isn't used in the article [4] because the analyze every trace separately and we are going to process all traces together. The first and second derivatives are obtained for the x-coordinate point and for the y-coordinate point. Finally the curvature is calculating using both derivatives for x-coordinate and for y-coordinate.

With this feature extraction method we got 8 features for every point in a trace. We have different trace length so we need an statistical model which can deal with dimension variations and we will see the proposed model later. We decide to use this approach in order to increase the size of the representation and with it the information. The 8-directional axes method [5] [7] simplify the representation using a discrete space. With our method instead of bounding it, we are adding more information to every tuple of points.

## 3.2 Offline data

To obtain the offline data from the online data, we draw a straight line between the points of every consecutive pair of tuples. With this we obtain an image representing the handwritten character. Our characters will be a matrix of pixels in gray scale, that is, we will have an integer value between 0 and 255 for every point in the image. In computer vision history have been developed many techniques to extract features from an image. Maybe the most used some years ago is to obtain SIFT descriptors [35] or the Fisher Kernel [38]. In our case we will feed the statistical model with the raw image. Our feature extraction for images will consist on obtaining the smallest bounding box where all the black pixels can be contained with a some padding to make the width equal to the height and then re-size this bounding box to a fixed size. In the next section we will see how our statistical model can deal with the raw image. We decided to use the raw image instead of using other feature extraction method because we want to be able to classify in a reasonable time.

# 4. Statistical model: Neural networks

To solve our classification task we need a statistical model which takes the feature vector and returns a score for every class. The score is necessary to short the classes and return the n best predictions. This model also must be able to deal with high number of classes due to the amount of Unicode characters and predict in a reasonable amount of time. We choose neural networks because we can found examples in the literature where the neural networks deal with a big number of classes [22][16]. The neural networks can also retrieve a score for every class if we use one hot encoding. They can predict in an acceptable time if the topology of the network are small enough and we take advantage with it parallelism capabilities.

In the next sections we are going to talk about some different neural network structures that we will use for our experiments. We will start with the multi layer perceptron (MLP). Then we are going to introduce the convolutional neural networks (CNN) with its convolutional and pooling layers. To end with the neural network structures we are going to talk about recurrent neural networks along with a special layer called long short term memory (LSTM) and a special way to stack two of them to create a layer called bidirectional long short term memory (BLSTM). To end with the chapter we will briefly explain how they learn the discriminant function and some resources to prevent the overfitting.

## 4.1 Multi Layer Perceptron (MLP)

The perceptron [39] is a statistical model which learns a linear discriminant function combining a set of weights with the feature vector. The multi layer perceptron (MLP) is a combination of several perceptrons along with nonlinear activation functions in order to be able of learn nonlinear discriminant functions. A MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Each node is called neuron and its followed by a nonlinear activation function except the input layer.

## 4.2 Convolutional Neural Network (CNN)

CNNs have been used as early as the nineties to solve character recognition tasks [29]. In CNNs we have got convolutional layers along with some optional pooling layers. In convolutional layers instead of having a weight for every input we have a set of filters called kernels, every kernel has a list of weights. The convolution operator between a image I and a kernel K is the result of applying the convolution to every pixel where the center of the filter can be position and don't get out of the boundary of the image. The convolution operator between a window and a kernel with the same dimensions consist on the sum of the element wise multiplication between the window and the kernel.In the Figure 4.1 we can see an

example. We can add some padding to the edges of the original image to avoid the dimensionality reduction after applying the convolution. The padding could take different values, in our case we are going to extend the image with the same values that we got in the edges.

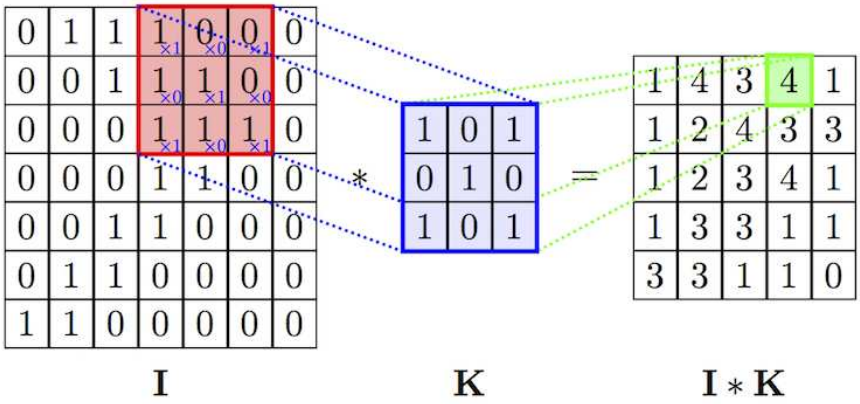


Figure 4.1: Convolution operator [55]

The max pooling operator decreases the dimension of your data simply by taking only the maximum input from a fixed region. As the convolutional layer this layer acts with a sliding window of a fixed size but instead of multiply the values it takes the biggest in the window. In Figure 4.2 we have an example of a max pooling layer with a kernel size of 2 x 2 and a stride (distance that we move the window to the next step) of 2. In practice the pooling layer not only helps to reduce the dimensionality, it reduces in a small factor the shift variability between samples.

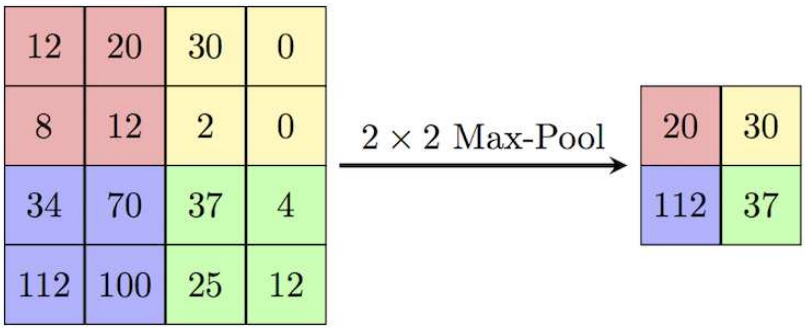


Figure 4.2: Maxpooling operator [55]

A convolutional neural network (Figure 4.3) is the combination of several convolutional and pooling layers in the same way that we do with the multilayer perceptron. It is common to add some fully connected layers at the end to adjust the dimensionality to the output target. Convolutional neural network architectures are not just used in image tasks [50] [18], they are also used in some different task like tweet sentiment classification [45], speech recognition [2] and some other task [31][23].

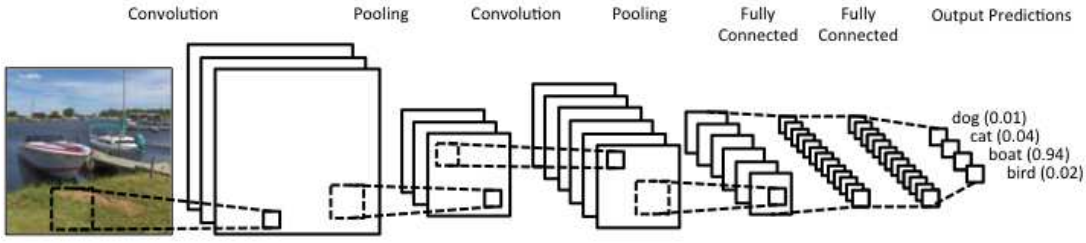


Figure 4.3: Convolutional neural network example [47]

## 4.3 Recurrent Neural Networks (RNNs)

Every neural network which has one or more recurrent layers is called recurrent neural network (RNN). This layers are normally used for sequences or some data with time dependencies. We can see in Figure 4.4 a example of a recurrent layer whit dimension one. In a recurrent layer the output in a time t will be used in t+1 multiplied by a weight which will be learning as all the other weights. We can see the layer unfold in three instants of time.

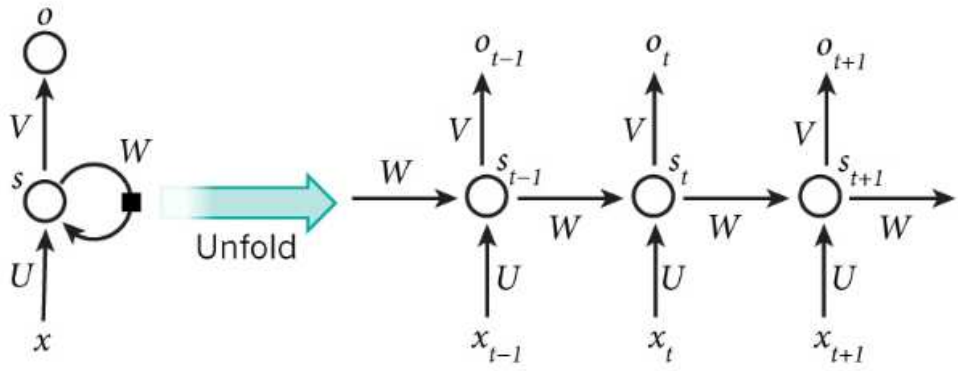


Figure 4.4: Recurrent layer example [8]

The idea behind RNNs, introduced in the decade of 1980 in [42], is to make use of sequential information. In a fully connected or a convolutional neural network we assume that all inputs (and outputs) are independent of each other. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps due to the vanish gradient problem. We are going to use this network to solve the difference of length between different characters with online features.

### 4.3.1 Bidirectional Recurrent Neural Networks

Bidirectional RNNs [44] are based on the idea that the output at time t may not only depend on the previous elements in the sequence, but also future elements.

Then, we will have two recurrent layers, the forward layer which explores the data from left to right and the backward layer which explores the data from right to left (Figure 4.5). For example, to predict a missing word in a sequence you want to look at both the left and the right context. Bidirectional RNNs are quite simple. They are just two RNNs stacked on top of each other. The output is then computed based on the hidden state of both RNNs.

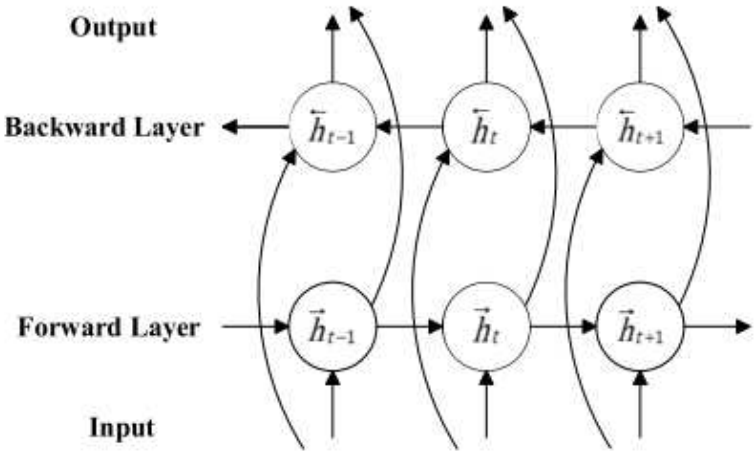


Figure 4.5: Bidirectional recurrent layer example [15]

### 4.3.2 Long Short Term Memory (LSTM)

The LSTM [19] networks don't have a fundamentally different architecture from RNNs, but they use a different function to compute the hidden state. The memory in LSTMs are called cells and you can think of them as black boxes that take as input the previous state h(t-1) and current input x(t). Internally these cells (Figure 4.6) decide what to keep in (and what to erase from) memory. They then combine the previous state, the current memory, and the input. It turns out that these types of units are very efficient at capturing long-term dependencies avoiding the gradient vanish problem.

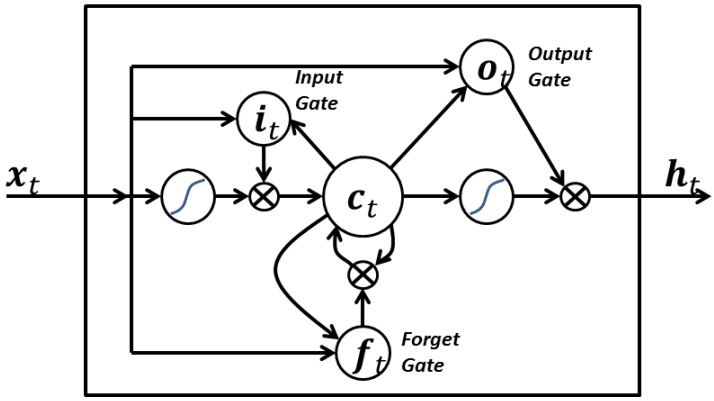


Figure 4.6: Long Short Term Memory cell [10]

As we said in the previous section we can have two recurrent layers, the forward layer which explores the data from left to right and the backward layer

which explores the data from right to left and instead of having simple recurrent layers we can substitute it for LSTM. This networks are called Bidirectional Long Short Term Memory (BLSTM).

## 4.4 Learning process

In our classification task the objective of the learning process is to find a set of weights that map any input to its correct output. This is in an optimal solution. In order to approximate the weights to the optimal we are going to learn in a supervised way. It means that we train the neural network with both the input and the expected output. Then it is possible to calculate the error based on the target's output and the calculated one. Then the difference between both (gradient) is used to update the weights using the back-propagation algorithm [41][40].

To avoid our classifier fits too much to the training data (overfitting) we have found in the literature some techniques with reduce it. One of the simplest and basics is to shuffle the training set every epoch, this makes the gradients of the back-propagation changes and learn the classes in different order every time. Other two techniques we are going to describe, that are recent and no so simple, are Dropout [48] and batch normalization [21].

### 4.4.1 Dropout

The dropout [48] is a technique which consist in simulate malfunction in some neurons. This force the neurons to learn a more general function due to in every train batch we change the neurons which are disabled. In training phase we decide a percentage of neurons which will be disable in every layer. Then randomly neurons disable its output. In test phase all the neurons will be active so we multiply the output of a layer by the percentage of dropout. In the Figure 4.7 we can see an example of a network in training phase.

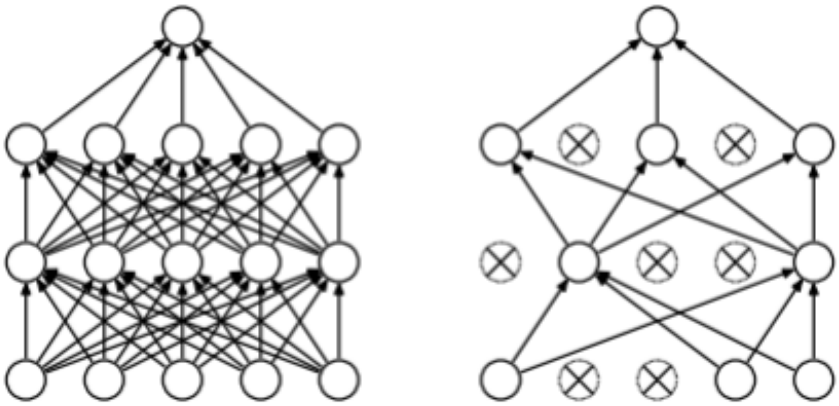

Figure 4.7: Dropout representation [48]

### 4.4.2 Batch normalization

Batch normalization [21] is an operation usually made to the output of a layer before the activation function. It is made to solve the covariate shift problem. Covariate shift is the change in the distribution of the covariates specifically, that is, the independent variables. This is normally due to changes in state of latent variables, which could be temporal (even changes to the stationary of a temporal process), or spatial, or less obvious. Batch normalization also helps to avoid overfitting due to it acts as a regularization operation too.

Batch normalization is based on the idea that we train our neural network with mini-batches (group of samples together) instead of sample by sample. In the recent frameworks for neural networks (like Tensorflow [1] or Theano [51]). Batch normalization operator uses the median and the variance of the actual batch to normalize the data. Batch normalization was firstly thinking for fully connected layers but in some recent publications we can see it used in convolutions [18]. And some recent studies applied it to recurrent neural networks [28] [11].

# 5. Experiments

To perform our experiments first we need a dataset. The dataset we need must be based on handwritten characters with every sample annotated at stroke level, that means that the dataset must have the path followed by the pen instead of the image, so we can extract online features and at the same time we can draw the character to extract offline features. The chosen dataset is UJI PEN [34] and we will describe it in the next section. In order to define our models we divide them in 3 categories. The first one is based on recurrent neural network to deal with variability in sample length when we use the online features. The second one uses fully connected and convolutional layers to recognize the offline images. And the last one uses a combination of previous models to take advantage of the two feature extraction methods.

All these models was programmed in Python using the library Keras [9]. Keras is a high-level neural network API, written in Python and capable of running on top of either TensorFlow [1] or Theano [51]. In our case we used Theano as back end for easy setup with CUDA [37] to be executed in a graphics processing unit (GPU).

## 5.1 Dataset: UJI PEN

The UJI pen character database is a set of handwritten character sample with every sample annotated at stroke level. The database has two versions. The first version (Figure 5.1) consist on a set of 26 ASCII letters (lower and uppercase) and 10 digits (from 0 to 9) written by 11 writers, from Universitat Jaume I (UJI). The classification task is a 35-class one because they have not considered a different class for each different character: each one of the 26 letters is considered as a case-independent class, there are 9 additional classes for non-zero digits, and the zero is included in the same class as o's. The handwriting samples were collected on a Toshiba Portégé M400 Tablet PC using its cordless stylus. Only X and Y coordinate information was recorded along the strokes by the acquisition program, without pressure level values or timing information.

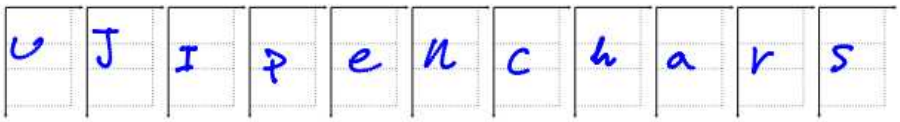

Figure 5.1: UJI pen v1 characters example [34]

The second version is an extension of the first adding 49 writers, carried out at Universitat Politècnica de València (UPV) with 44 writers and UJI with 5 writers. These dataset consist on:

- The 52 ASCII letters, including uppercase and lowercase letters.

- 14 Spanish non-ASCII letters: ñ, Ñ, vowels with acute accent (10 characters), ü and Ü.
- The 10 digits from 0 to 9.
- Punctuation and other symbols(. , ; : ? ! ’ ’ ( ) % - @ <>$ ¿ ¡ ♂ ♀ €)

In total we have 97 classes with 120 samples per label (11640 samples in total). Our approach consists on divide the dataset in 50 % for training, 5 % for validation and 45 % for testing. We divide our generation in two, the first one using distortions and the second one using variational autoencoder (VAE). With the first one we generate 18 new samples for every original so we obtain a total of 1080 samples for label, 104760 for training. With the VAE technique we generate 464 samples for class obtaining 45008 samples in total. The generation method used in first places include the original samples in the set, the VAE technique doesn't include the original samples. In Table 5.1 we have a summary of how many samples we got for every set including validation and test partitions.

Table 5.1: Number of samples for set

| | **training original** | **VAE** | **VAE + original** | **Distortions** | **Validation** | **Test** |
|---|---|---|---|---|---|---|
| **number of samples** | 5820 | 45008 | 50828 | 104760 | 582 | 5238 |

## 5.2 Online features

As we said before we will use recurrent neural network to deal with variability in sample length. The neural networks we are going to use in this section are compose by a recurrent layer which receives the features as input, stack all the information in an array and process it using fully connected layers. The problem with the variability in sample length is that we only can train one sample at a time instead of a group of samples at a time (batch). In order to reduce the training time we group the samples with the same length which allows us to make batches at the same length. This technique is known as bucketing [24]. In previous section we mention the generation of synthetic samples in order to improve the generalization of our classifier, in this section we don't have and easy way to create new labeled samples so we only use the real samples to train our neural networks and see how good we can classify.

### 5.2.1 Long Short Term Memory (LSTM)

Our first approach with recurrent neural networks was the LSTM, and we tried three different topologies. These three neural network structures are shown in Figure 5.2. We start from a very basic topology and try to improve it with a few changes.
The first one (the model at the left) is a very simple neural network with one LSTM layer with 128 neurons which condense all the time information in a 128

dimensional vector followed to it 2 more hidden layers with 256 and 128 neurons respectively with Relu activation.
The second one is based on the previous one. The only change that we did was adding another LSTM layer after the first LSTM, so the layer who will resume the time information will be the second one instead of the first.
The third and last topology we tried is the model 3, it is based on the second topology but we duplicated the number of neurons in the second LSTM layer and in the two fully connected layers.

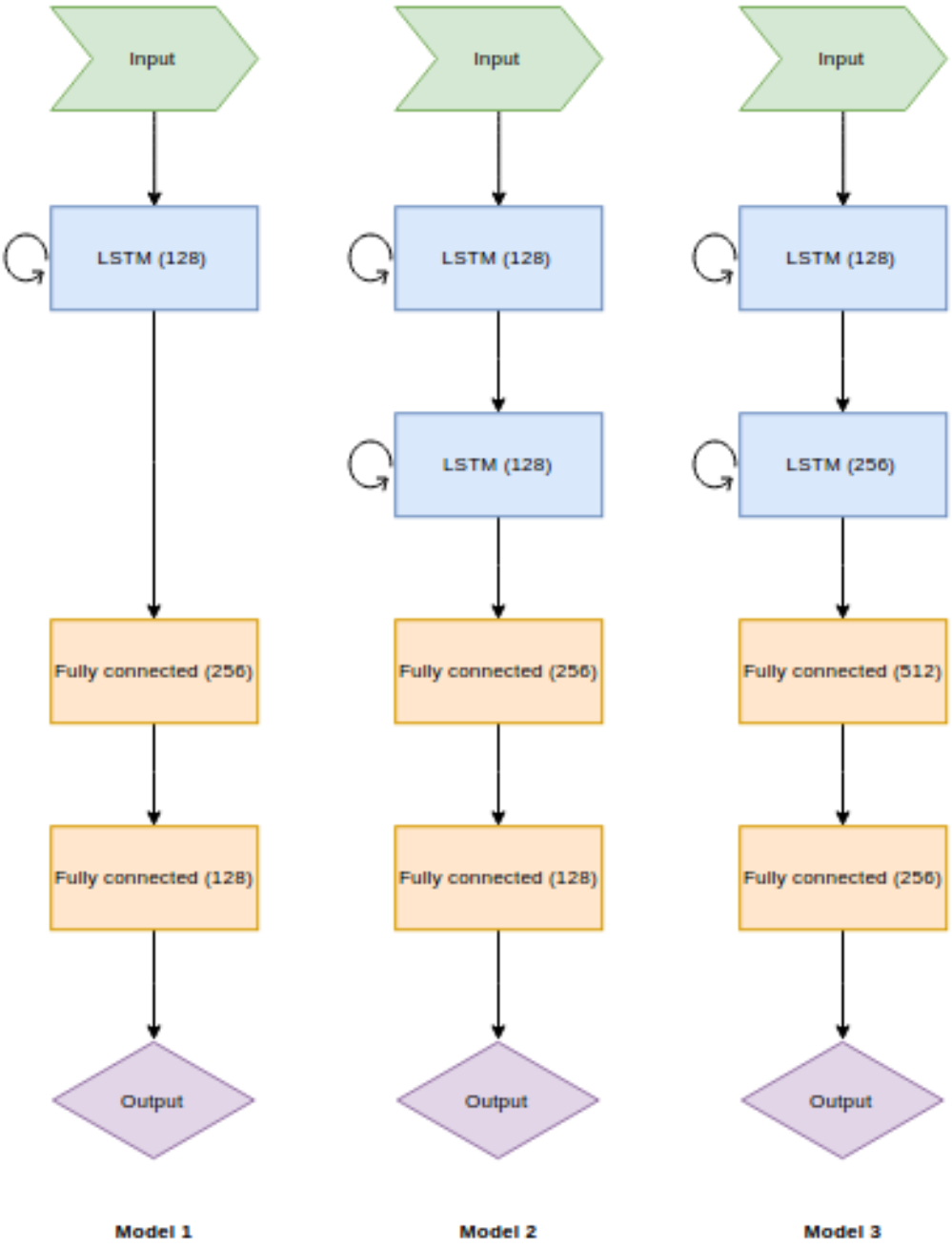


Figure 5.2: LSTM models

In Table 5.2 we shown the results of training the three previous models with the original data partition for train and evaluate its accuracy with the test parti-

tion. All the models have been trained the same number of epochs (400). All the fully connected layers had batch normalization applied and a dropout coefficient of 20 %. We choose the epoch with the best accuracy obtained in the validation set and then evaluate with the test set, the error percentage with the test are written in the table.

Table 5.2: LSTM results

| | **Error** |
|---|---|
| **Model 1** | 62,39 % |
| **Model 2** | **30,09** % |
| **Model 3** | 34,53 % |

As we can see adding a new LSTM layer improves a lot the results but increasing the dimensions of the layers doesn't improve the accuracy. As we can see increasing the size of the neural network not always improve the classifier, but we are talking with a limited number of epochs due to the time of execution, a bigger neural network has more parameters and need more time to learn the optimal values for every one. In the next section we will compare the use of a simple LSTM layer with the use of two of them as a layer called bidirectional long short term memory, one to read the features from left to right and other from right to left.

### 5.2.2 Bidirectional Long Short Term Memory (BLSTM)

In order to try to improve our last results we repeat the experiment with BLSTM instead of LSTM. As we can see in Figure 5.3 the topologies are the same as in the previous section but instead of using LSTM we use BLSTM.

As we did in the previous section, we train for only 400 epoch using batch normalization and a 20 % of dropout in the fully connected layers. In Table 5.3 we shown the best results obtained in one of these 400 epoch for every model along with the previous results of the LSTM in order to see the difference between them. As we say before, a bigger neural network requires more epochs and data to train but due to the time limitations and the neural network training speed we limited the epochs to 400 as we do in the previous section to make a fair comparison.

Table 5.3: BLSTM results

| | **LSTM error** | **BLSTM error** |
|---|---|---|
| **model 1** | 62,39 % | 61,40 % |
| **model 2** | **30,09** % | **28,76** |
| **model 3** | 34,53 % | 35,66 |

The results shows a little improvement in two of the three models, as we said before, it may be due to the insufficient number of epochs along with the size of the corpus. The model 2 using BLSTM will be used later to combine it with a classifier which take care of offline features to try to improve our results.

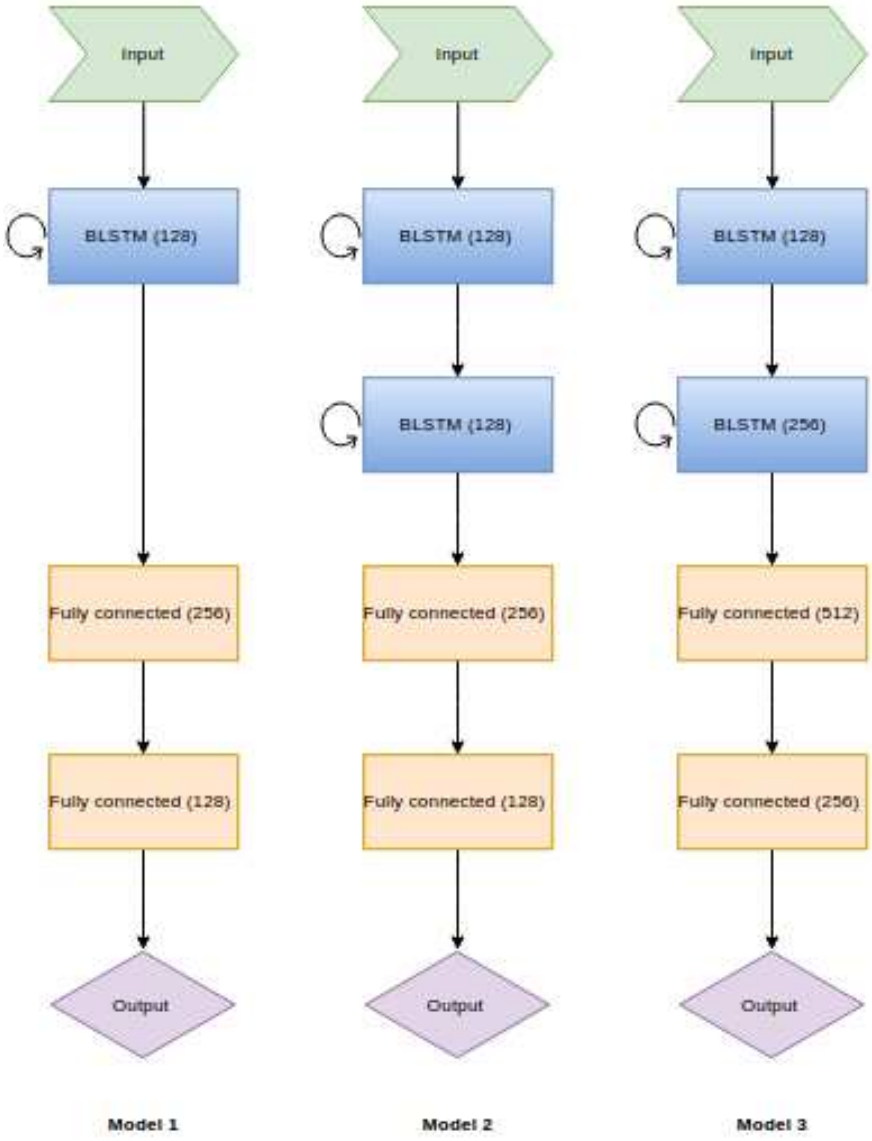


Figure 5.3: BLSTM models

## 5.3 Offline features

In this section we are going to use the raw image, every pixel will be one feature, so our feature vector will be of 4900 features (70x70 pixels). We are going to compare multilayer perceptron (only fully connected layers) with convolutional neural networks. In this section we are going to compare the two generation methods, the distortion one with the variational autoencoder (VAE) one. As we said before we have 5820 original samples, 45008 generated with the variational autoencoder (VAE) and 104760 generated with the distortions.

### 5.3.1 Multi Layer Perceptron (MLP)

Before feeding our neural network with our images we transformed the image in a linear vector (from a 70x70 matrix to a 4900 vector), and then it was normalized with mean 0 and a variance of 1. In Figure 5.4 are shown the three topologies

which we experimented. All of these models are formed by three hidden layers along with batch normalization and a 20 % of dropout rate for every one. We train every model 150 epochs with four different set of data, only the original ones, only the generated with the VAE method, the originals besides the generated with the VAE and finally the generated with distortions.

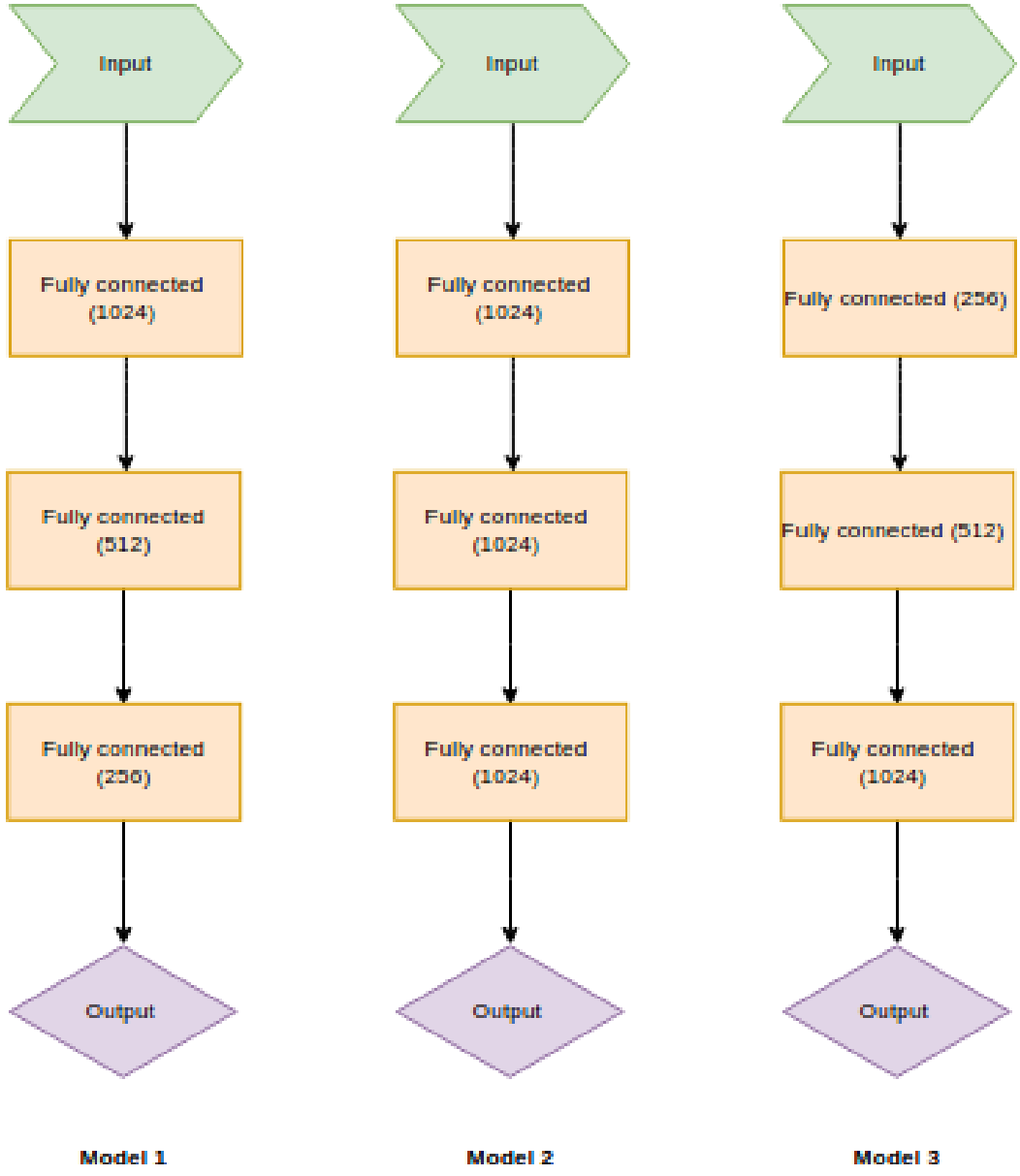


Figure 5.4: MLP models

In Table 5.4 we have got the results of the experiments. We can see that the generation of synthetic samples improve our models accuracy. In our case the generation with distortions far exceeds the results the generation with the VAE along with the originals. As we said before not the biggest model (model 2) achieve the best accuracy, in this case the model 3 with the data generated with distortions gets the best score even when the model 2 obtain a best accuracy with only the original data.

Table 5.4: MLP results

| Model | Error | | | |
|---|---|---|---|---|
| | **Originals** | **VAE** | **VAE and originals** | **Distortions** |
| **Model 1** | 45,57 % | 62,56 % | 50,4 % | 26,69 % |
| **Model 2** | 39,39 % | 66,25 % | 47,4 % | 30,47 % |
| **Model 3** | 45,53 % | 68,33 % | 49,04 % | **25,49 %** |

### 5.3.2 Convolutional neural network (CNN)

The experiments with convolutional neural networks followed the same direction as the previous. We tried three different models. Unlike the previous sections we tried more complex models as discussed below. The models was trained 150 epochs using batch normalization in both layers, convolutional and fully connected.

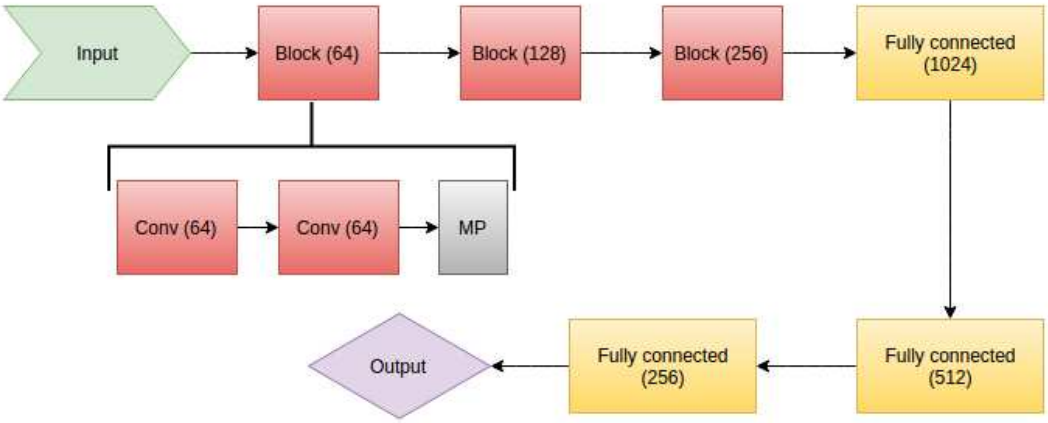


Figure 5.5: CNN model 1

The first model we tried is shown in Figure 5.5. It is composed of 3 block of convolutions, each block have two consecutive convolutional layers and a max pooling layer. Each block has a parameter which indicates the number of kernels used in the convolutional layers, the size of these kernels is fixed on a size of 5x5 and the max pooling layer uses a 2x2 window and a stride of 2.

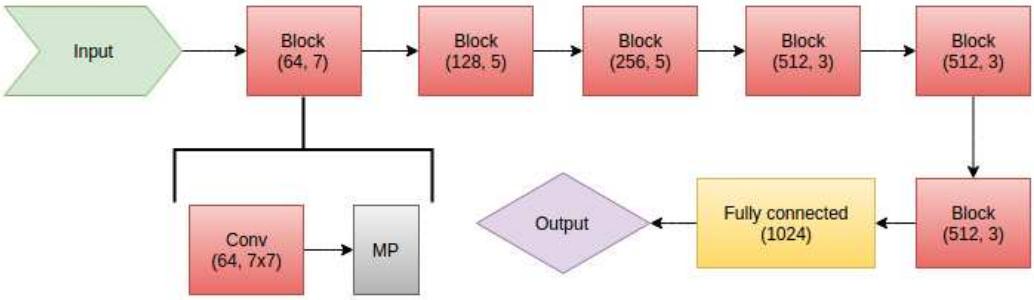


Figure 5.6: CNN model 2

The second model is represented in Figure 5.6. In this case we have 6 block

of convolutions, each block is constituted with a convolutional layer and a max pooling layer. Every block has two parameters, the first one is the number of kernels used in this convolution and the second one the height and width of the kernels (the kernels we used are squared). Every max pooling layer uses a window of 2x2 and a stride of 2.

The third model we used is based on GoogLeNet [50], this is a complex model which was the winner in 2014 of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The neural was called inception due to the topology of the network along to the way to join convolutional layers in the same level. In Figure 5.7 we can see how it is represented a block in this network. The basic idea is to filter the output of the previous layer with three different convolutional layers along with a max pooling layer and concatenate it's outputs. We made some changes to the original network to adapt it to the size of our data.

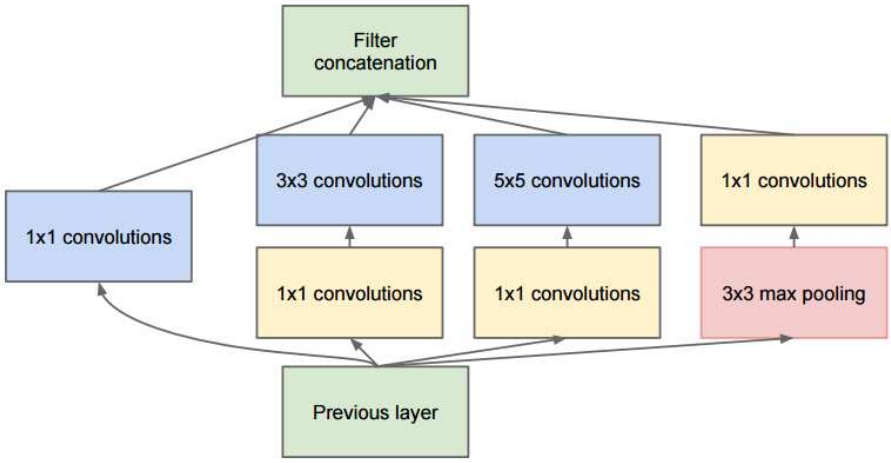


Figure 5.7: Inception block [50]

In table 5.5 we can see the architecture for our inception model. We can see in every row the layer (or block) that we are using in this depth level. In the second column the sizes of the kernels along with the strides. In third column we have the output size of this layer in height x width x channels. The other columns are specifically for the inception blocks, for every row the number is the amount of kernels used and the column title is the kernel size except if the have reduce after, in this case it reefers to the 1x1 convolutional layer before the nxn layer. In the case of pool column it is the number of kernels used in the 1x1 convolution before 3x3 max pooling. Every convolution has got padding to avoid modifying the size. The last layer is a fully connected layer with a softmax activation.

As it is said in the article [50], we use two auxiliary outputs, one after the inception block 4a and another after the inception block 4d shown in Table 5.5. This auxiliary outputs have the same topology. First an average pooling with a windows size of 5x5 and stride of 3. Then a convolutional layers with 128 kernels of 1x1. After that a fully connected layer with size of 1024 and a dropout rate of 70 % followed by another fully connected layer with the same size as the number

Table 5.5: CNN model 3 architecture

| **type** | **patch size/ stride** | **output size** | **1x1** | **3x3 reduce** | **3x3** | **5x5 reduce** | **5x5** | **Pool** |
|---|---|---|---|---|---|---|---|---|
| convolution | 5x5/1 | 70x70x64 | | | | | | |
| max-pool | 2x2/2 | 35x35x64 | | | | | | |
| convolution | 3x3/1 | 35x35x192 | | | | | | |
| inception (3a) | | 35x35x256 | 64 | 96 | 128 | 16 | 32 | 32 |
| inception (3b) | | 35x35x480 | 128 | 128 | 192 | 32 | 96 | 64 |
| max-pool | 2x2/2 | 17x17x480 | | | | | | |
| inception (4a) | | 17x17x512 | 192 | 96 | 208 | 16 | 48 | 64 |
| inception (4b) | | 17x17x512 | 160 | 112 | 224 | 24 | 64 | 64 |
| inception (4c) | | 17x17x512 | 128 | 128 | 256 | 24 | 64 | 64 |
| inception (4d) | | 17x17x528 | 112 | 144 | 288 | 32 | 64 | 64 |
| inception (4e) | | 17x17x832 | 256 | 160 | 320 | 32 | 128 | 128 |
| max-pool | 2x2/2 | 8x8x832 | | | | | | |
| inception (5a) | | 8x8x832 | 256 | 160 | 320 | 32 | 128 | 128 |
| inception (5b) | | 8x8x1024 | 384 | 192 | 384 | 48 | 128 | 128 |
| avg-pool | | 1x1x1024 | | | | | | |
| dropout | 40% | 1x1x1024 | | | | | | |
| linear | | 1x1x5488 | | | | | | |

of labels with a softmax activation function. This two auxiliary outputs helps the network to learn more easily the weights at the beginning.

In Table 5.6 are shown the results of the experiments with the CNN's models besides the results with the MLP's models. As we can see, all the convolutional neural network improve the accuracy obtained by the MLP's. In our future model to classify Unicode characters we will use convolutional networks to deal with the offline data. In the next section we are going to use both data online and offline to see if adding information helps to the classification.

Table 5.6: CNN and MLP results

| **Neural network** | **Model** | **Accuracy** | | | |
|---|---|---|---|---|---|
| | | **Originals** | **VAE** | **VAE and originals** | **Distortions** |
| **Multilayer perceptron** | **Model 1** | 45,57 % | 62,56 % | 50,4 % | 26,69 % |
| | **Model 2** | 39,39 % | 66,25 % | 47,40 % | 30,47 % |
| | **Model 3** | 45,53 % | 68,33 % | 49,04 % | **25,49** % |
| **Convolutional** | **Model 1** | 19,66 % | 36,48 % | 23,35 % | 18,98 % |
| | **Model 2** | 18,21 % | 40,76 % | 22,17 % | 15,75 % |
| | **Model 3** | 16,35 % | 29,95 % | 19,64 % | **15,33** % |

## 5.4 Combining features

In this section we are going to use both, online data and offline data together to classify. We are going to try two ways to use this information. The first way is to make a linear combination of the best classifier for every type of feature, model 2 of the BLSTM's and model 3 of the CNN's. The second way is to create a neural network which accepts two inputs, first input will take the online features and the second input the offline features and train it like one classifier.

### 5.4.1 Linear combination

Every classifier we had trained had as the output the posterior probability thanks to the softmax activation function and the one hot encoding. We can make a linear combination of the probability of one model with another. As we have the two classifiers trained. The first one is the model 2 using BLSTM's trained only with the original data with the online feature extraction method. The second one is the model 3 (inception model) using CNN's, we are going to use the model trained with the data created using the distortions due to it's good performance.

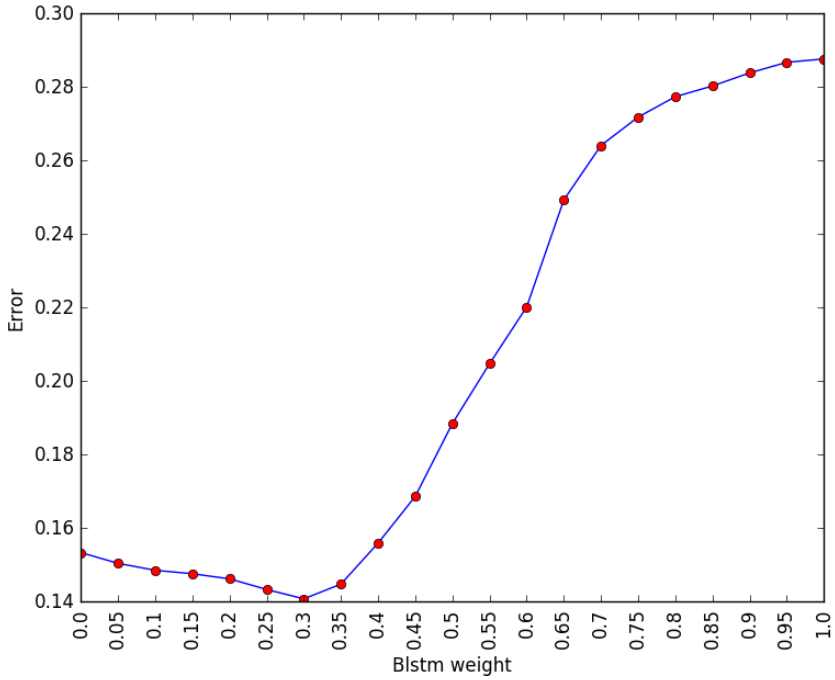


Figure 5.8: Linear combination results

We have plotted the error in Figure 5.8. We made several experiments varying the weights for every model. In the x-axis we have the value of the weight for the BLSTM model, for the CNN model the weight will be 1 less the weight of the BLSTM. The best result obtained was with a weight of 0.3 for the BLSTM (0.7 for CNN) where the error was a 14,08 %. The best result obtained with CNN was 15,33 % and with the BLSTM was 28,76 % so we can say that the linear combination improves the results if we found the perfect weight for every one and in any case the result are worse than the baddest model alone.

### 5.4.2 Early fusion

In this case instead of using the two classifiers we are going to join the two feature extraction methods in one. To deal with it we need a neural network which accepts two inputs, first input will take the online features and the second input the offline features. In Figure 5.9 we can see the neural network that we used for this experiment.

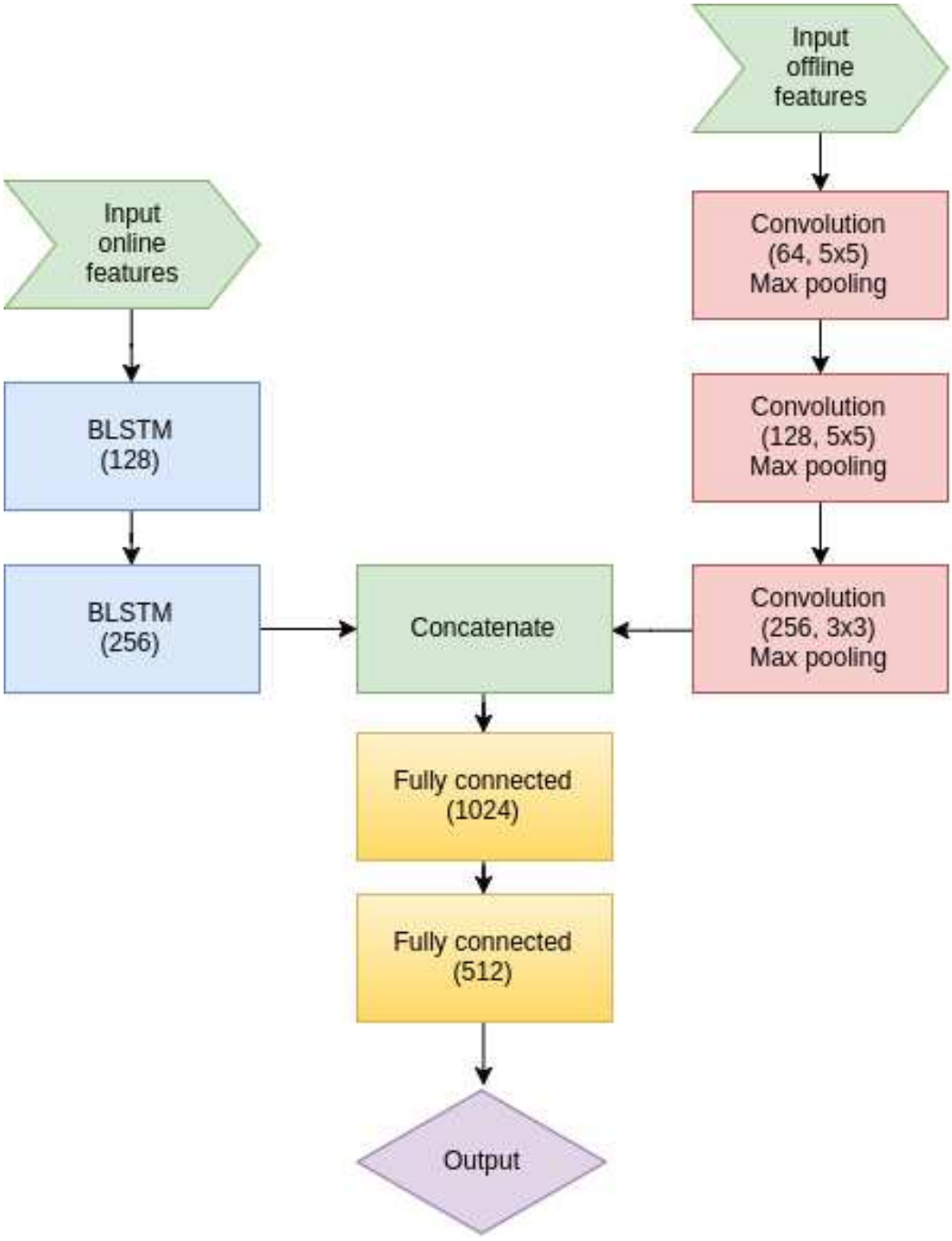


Figure 5.9: Two inputs neural network

The network joins two basic approaches, in one hand we have tho BLSTM's which take care of the online features, in the other hand we have three blocks,

each block is formed by a convolutional layer follow by a max pooling layer. The output of this two path joins in a flat vector which feeds two fully connected layers. This network has been trained 150 epoch. Every fully connected layer have applied batch normalization and have a dropout coefficient of 20 %.

Table 5.7: Combining features results

| | **Best accuracy** |
|---|---|
| **Online features** | 28,76 % |
| **Offline features** | 15,33 % |
| **Linear combination** | 14,08 % |
| **Two inputs neural network** | 25,63 % |

In Table 5.7 are shown the best result obtained in every category. Only using the online features, only using offline features, with the linear combination and with the neural network with both types of features. As we can see the best result was obtained with the linear combination and the worse with only the online features. We can say that in this experiment adding the online features don't help our neural network to improve this can be due to we add more parameter to be stimated and we don't increase our number of training samples.

# 6. Demo

In order to make easy the testing of our classifier, we developed a demo hosted in http://transcriptorium.eu/demots/unicode/demo.html. Our demo is a simple web page (Figure 6.1) where we can draw a character and submit it to the server to get the n nearest Unicode characters. We tried two approaches to return the solutions. The fist one is to return a constant number of solutions, we consider 10 solutions a good number. the second one is to return solutions till the sum of the scores, in our case the posterior probability, is greater than a threshold. We choose to use the first method due to sometimes where the character classification is not so precise the probabilities are so small that we return a big number of solutions.

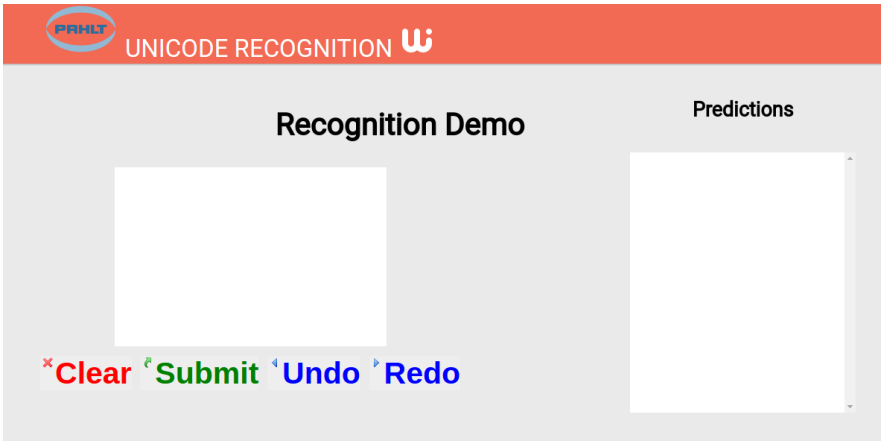


Figure 6.1: Demo web page

In the case of the classifier model we choose to use offline data along with the model 2 (Figure 5.6) of convolutional neural networks. We could choose the model 3 which achieved best accuracy on the UJI Pen dataset but we decided to use the the model 2 due to the time of predictions. The model 2 can classify a sample in less than 2 seconds and the model 3 needs more than 7. Both models where tested in CPU because the server where the demo is hosted don't have a GPU.

Figure 6.2: Example of predictions

In Figure 6.2 we got some examples of real characters classified. In the first column we have got the characters that were drawn in the demo, before classify

them we extract the minimal bounding box which contains the character and resize it to our predefined dimensions with the same aspect ratio. From the second till the last column we got the predictions ordered from most probably to less.

# 7. Conclusion

In section 1.2 we previously define four objectives which guides our methodology to finally develop the demo and in the future the final system. The objectives along with its proposed solution are the next:

- In first place we have to obtain a set of labeled handwritten samples which contains all of our characters. We tried two different ways to generate data, the first one using distortions to the original data and the second one using a variational autoencoder. We used the first technique due to its good results and the second needs some more data than the originals. We can use both together but we will discuss in the section Future work.

- In second place we have to find a good representation which we can train our neural network. In this case we have to choose to representation, one for online data and another for offline data. In the case of online data we choose use the derivatives along with the curvature. For the offline data we use the normalized image as the representation. As we said in the section Experiments the models which gets the offline features as the input gets better results than the ones which takes the online features.

- In the third objective we have to choose the statistical model which deal with the data. We choose neural networks which in our experiments don't have problems with the big number of classes and return the posterior probability for every class given a sample. So the n best solutions proposed by our model will be the n with higher probability. To measure how well work this approach along with the feature extraction we choose we used the UJI PEN [34] character database. The best result was obtained with a liner combination between the best model which takes online data and the best which takes offline data.

- The last objective is to scale the model to classify in a bigger set of characters. We didn't talk about this before, and we will discuss later on next section 8.2. We will propose some approaches to include the whole Unicode set. The most easy way to include them is to substitute the last layer in the neural network with one of the size of the new number of classes and retrain the network taking advantage of the fact that the rest of the layers are pretrained.

# 8. Future work

In the problem we are dealing with, there are two main areas where we can improve. This areas are the data and the models. In the data section (8.1) we will describe the work we could do to improve our data quantity and quality. In models section we will deal with the issues of scalability and precision.

## 8.1 Data

The first step to increase our data quantity is the logic step, generate more. The simple way to do this is to introduce new distortions in our generation. One of these distortions could be a four point perspective transform (Figure 8.1). Where we simulate to see the character from a different point of view.

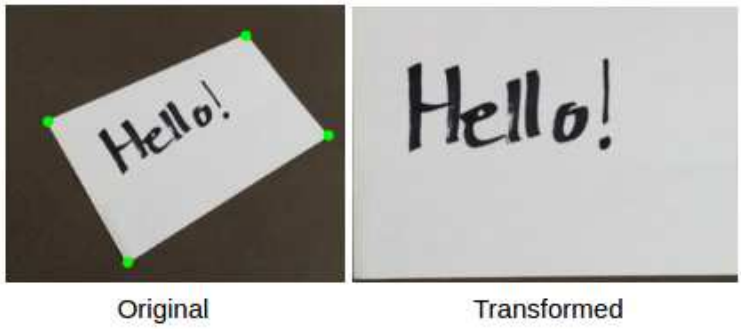


Figure 8.1: Four point perspective transform

Another way to increase our data generation is to join our two proposal methods, the distortion approach and the variational autoencoder (VAE) one. First we can do a base generation with the distortion method and the with the generated samples we can use our VAE to generate more from every pair of samples with the same label. These generation could also be done on the fly during the neural network training.

In favour of increasing our data quality we can make an acquisition of real samples with the tool that we developed. This collection can be small and then be increased with our distortion method or with the VAE. We also could save the stroke information, extract its online features and make a generation of online features with the VAE using some recurrent layer which can deal with it.

## 8.2 Models

We will center in this section in two features to improve, precision and scalability. The precision of our model can't be measured in an objective way due to we haven't got a real set to test our model, we only can measure it subjectively making test by our own with a few characters. We saw that the chosen model is based on a state-of-the-art classifier for ImageNet in 2014 [12] [50]. We can

improve our model with the use of residual networks [18], densely connected residual networks [20], residual of residual networks [58] or even the new version of GoogLeNet [49] which uses residual connections along with its inception model. Another improvement can be achieved if we make an acquisition of real samples an we use the strokes to obtain online features and we combine them with the offline features as we shown in the section 5.4.

The second area is scalability. As we said in the beginning, the recent Unicode character set contain more than 128,000 characters, in our case we are only using less than 5,500. The logic step is to repeat all the process we make with the whole Unicode character set, but it will be slow and if the Unicode set increase this model can't scale to accept them all. Another way is to substitute the last layer with another fully connected with the size of the number of classes and train the whole network with a few epoch taking advantage of the other layers of the network that are trained with the small set. There is other alternatives, the one we would choose is to use our actual neural network as a feature extractor for images. We could cut our neural network near to the end and get this layer output as a feature vector. Then we can use another algorithm which can be easily scalable like k nearest neighbour to classify our new samples. To increase our character set which can be recognized we only have to add new classes samples to the search space. Another approach is to use one classifier to every class, this classifier predict the probability that a sample is from this class. To add new classes we only need to train new classifiers.

# Bibliography


[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.

[3] J. Aliprand. *The Unicode standard*. Addison-Wesley, Boston, 2003.

[4] F. Álvaro, J.-A. Sánchez, and J.-M. Benedí. An integrated grammar-based approach for mathematical expression recognition. *Pattern Recognition*, 51:135–147, 2016.

[5] Z.-L. Bai and Q. Huo. A study on the use of 8-directional features for online handwritten chinese character recognition. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, pages 262–266. IEEE, 2005.

[6] S. Belongie, G. Mori, and J. Malik. Matching with shape contexts. *Statistics and Analysis of Shapes*, 1:3–1, 2006.

[7] U. Bhattacharya, B. K. Gupta, and S. Parui. Direction code based features for recognition of online handwritten characters of bangla. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 1, pages 58–62. IEEE, 2007.

[8] D. Britz. WILDMl: Recurrent Neural Networks tutorial. `https://cambridgespark.com/content/tutorials/convolutional-neural-networks` 2015.

[9] F. Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[10] W. Commons. File:long short term memory.png — wikimedia commons, the free media repository. `https://commons.wikimedia.org/w/index.php?title=File:Long_Short_Term_Memor` 2017.

[11] T. Cooijmans, N. Ballas, C. Laurent, Ç. Gülçehre, and A. Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[13] T. DeVries and G. W. Taylor. Dataset augmentation in feature space. *arXiv preprint arXiv:1702.05538*, 2017.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[15] A. Graves, N. Jaitly, and A. rahman Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *ASRU*, 2013.

[16] T.-L. Ha, J. Niehues, and A. Waibel. Lexical translation model using a deep neural network architecture. *arXiv preprint arXiv:1504.07395*, 2015.

[17] M. Hamanaka, K. Yamada, and J. Tsukumo. On-line japanese character recognition experiments by an off-line method based on normalization-cooperated feature extraction. In *Document Analysis and Recognition, 1993., Proceedings of the Second International Conference on*, pages 204–207. IEEE, 1993.

[18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[19] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[20] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[21] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[22] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman. Reading text in the wild with convolutional neural networks. *CoRR*, abs/1412.1842, 2014.

[23] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[24] A. Karpathy, J. Johnson, and L. Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[25] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[27] F. Lauer, C. Y. Suen, and G. Bloch. A trainable feature extractor for handwritten digit recognition. *Pattern Recognition*, 40(6):1816–1824, 2007. 19 pages.

[28] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio. Batch normalized recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 2657–2661. IEEE, 2016.

[29] Y. Le Cun, L. Bottou, and Y. Bengio. Reading checks with multilayer graph transformer networks. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, volume 1, pages 151–154. IEEE, 1997.

[30] Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[31] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[33] C.-L. Liu, F. Yin, D.-H. Wang, and Q.-F. Wang. Online and offline handwritten chinese character recognition: benchmarking on new databases. *Pattern Recognition*, 46(1):155–162, 2013.

[34] D. Llorens, F. Prat, A. Marzal, J. M. Vilar, M. J. Castro, J.-C. Amengual, S. Barrachina, A. Castellanos, S. E. Boquera, J. Gómez, et al. The ujipenchars database: a pen-based database of isolated handwritten characters. In *LREC*, 2008.

[35] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[36] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[37] NVIDIA Corporation. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA Corporation, 2007.

[38] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[39] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[40] F. Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961.

[41] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[43] R. Salakhutdinov and G. Hinton. Deep boltzmann machines. In *Artificial Intelligence and Statistics*, pages 448–455, 2009.

[44] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[45] A. Severyn and A. Moschitti. Unitn: Training deep convolutional neural network for twitter sentiment classification. In *SemEval@ NAACL-HLT*, pages 464–469, 2015.

[46] P. Y. Simard, D. Steinkraus, J. C. Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

[47] G. Sood. clarifai. `https://www.clarifai.com/technology`, 2017.

[48] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[49] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

[50] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[51] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[52] A. O. Thomas, A. Rusu, and V. Govindaraju. Synthetic handwritten captchas. *Pattern Recognition*, 42(12):3365–3373, 2009.

[53] T. Varga and H. Bunke. Effects of training set expansion in handwriting recognition using synthetic data. In *Proc. 11th Conf. of the Int. Graphonomics Society*, pages 200–203, 2003.

[54] T. Varga and H. Bunke. Generation of synthetic training data for an hmm-based handwriting recognition system. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pages 618–622. IEEE, 2003.

[55] P. Veličković. Cambridge Spark: Deep learning for complete beginners: convolutional neural networks with keras. https://cambridgespark.com/content/tutorials/convolutional-neural-networks 2017.

[56] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 1058–1066, 2013.

[57] Wiris team. Wiris webpage. http://www.wiris.com/.

[58] K. Zhang, M. Sun, X. Han, X. Yuan, L. Guo, and T. Liu. Residual networks of residual networks: Multilevel residual networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 2017.

[59] Z. Zhang, L. Jin, K. Ding, and X. Gao. Character-sift: a novel feature for offline handwritten chinese character recognition. In *Document Analysis and Recognition, 2009. ICDAR'09. 10th International Conference on*, pages 763–767. IEEE, 2009.

# A. Annexed I: List of Unicode characters

U0009, U000a, U000D, U0020, U0021, U0022, U0023, U0024, U0025, U0026, U0027, U0028, U0029, U002a, U002b, U002c, U002d, U002e, U002f, U0030, U0031, U0032, U0033, U0034, U0035, U0036, U0037, U0038, U0039, U003a, U003b, U003c, U003d, U003e, U003f, U0040, U0041, U0042, U0043, U0044, U0045, U0046, U0047, U0048, U0049, U004a, U004b, U004c, U004d, U004e, U004f, U0050, U0051, U0052, U0053, U0054, U0055, U0056, U0057, U0058, U0059, U005a, U005b, U005c, U005d, U005e, U005f, U0060, U0061, U0062, U0063, U0064, U0065, U0066, U0067, U0068, U0069, U006a, U006b, U006c, U006d, U006e, U006f, U0070, U0071, U0072, U0073, U0074, U0075, U0076, U0077, U0078, U0079, U007a, U007b, U007c, U007d, U007e, U00a0, U00a1, U00a2, U00a3, U00a4, U00a5, U00a6, U00a7, U00a8, U00a9, U00aa, U00ab, U00ac, U00ad, U00ae, U00af, U00b0, U00b1, U00b2, U00b3, U00b4, U00b5, U00b6, U00b7, U00b8, U00b9, U00ba, U00bb, U00bc, U00bd, U00be, U00bf, U00c0, U00c1, U00c2, U00c3, U00c4, U00c5, U00c6, U00c7, U00c8, U00c9, U00ca, U00cb, U00cc, U00cd, U00ce, U00cf, U00d0, U00d1, U00d2, U00d3, U00d4, U00d5, U00d6, U00d7, U00d8, U00d9, U00da, U00db, U00dc, U00dd, U00de, U00df, U00e0, U00e1, U00e2, U00e3, U00e4, U00e5, U00e6, U00e7, U00e8, U00e9, U00ea, U00eb, U00ec, U00ed, U00ee, U00ef, U00f0, U00f1, U00f2, U00f3, U00f4, U00f5, U00f6, U00f7, U00f8, U00f9, U00fa, U00fb, U00fc, U00fd, U00fe, U00ff, U0100, U0101, U0102, U0103, U0104, U0105, U0106, U0107, U0108, U0109, U010a, U010b, U010c, U010d, U010e, U010f, U0110, U0111, U0112, U0113, U0114, U0115, U0116, U0117, U0118, U0119, U011a, U011b, U011c, U011d, U011e, U011f, U0120, U0121, U0122, U0123, U0124, U0125, U0126, U0127, U0128, U0129, U012a, U012b, U012c, U012d, U012e, U012f, U0130, U0131, U0132, U0133, U0134, U0135, U0136, U0137, U0138, U0139, U013a, U013b, U013c, U013d, U013e, U013f, U0140, U0141, U0142, U0143, U0144, U0145, U0146, U0147, U0148, U0149, U014a, U014b, U014c, U014d, U014e, U014f, U0150, U0151, U0152, U0153, U0154, U0155, U0156, U0157, U0158, U0159, U015a, U015b, U015c, U015d, U015e, U015f, U0160, U0161, U0162, U0163, U0164, U0165, U0166, U0167, U0168, U0169, U016a, U016b, U016c, U016d, U016e, U016f, U0170, U0171, U0172, U0173, U0174, U0175, U0176, U0177, U0178, U0179, U017a, U017b, U017c, U017d, U017e, U017f, U0180, U0181, U0182, U0183, U0184, U0185, U0186, U0187, U0188, U0189, U018a, U018b, U018c, U018d, U018e, U018f, U0190, U0191, U0192, U0193, U0194, U0195, U0196, U0197, U0198, U0199, U019a, U019b, U019c, U019d, U019e, U019f, U01a0, U01a1, U01a2, U01a3, U01a4, U01a5, U01a6, U01a7, U01a8, U01a9, U01aa, U01ab, U01ac, U01ad, U01ae, U01af, U01b0, U01b1, U01b2, U01b3, U01b4, U01b5, U01b6, U01b7, U01b8, U01b9, U01ba, U01bb, U01bc, U01bd, U01be, U01bf, U01c0, U01c1, U01c2, U01c3, U01c4, U01c5, U01c6, U01c7, U01c8, U01c9, U01ca, U01cb, U01cc, U01cd, U01ce, U01cf, U01d0, U01d1, U01d2, U01d3, U01d4, U01d5, U01d6, U01d7, U01d8, U01d9, U01da, U01db, U01dc, U01dd, U01de, U01df, U01e0, U01e1, U01e2, U01e3, U01e4, U01e5, U01e6, U01e7, U01e8, U01e9, U01ea, U01eb, U01ec, U01ed, U01ee, U01ef, U01f0, U01f1,

U01f2, U01f3, U01f4, U01f5, U01f6, U01f7, U01f8, U01f9, U01fa, U01fb, U01fc, U01fd, U01fe, U01ff, U0200, U0201, U0202, U0203, U0204, U0205, U0206, U0207, U0208, U0209, U020a, U020b, U020c, U020d, U020e, U020f, U0210, U0211, U0212, U0213, U0214, U0215, U0216, U0217, U0218, U0219, U021a, U021b, U021c, U021d, U021e, U021f, U0220, U0221, U0222, U0223, U0224, U0225, U0226, U0227, U0228, U0229, U022a, U022b, U022c, U022d, U022e, U022f, U0230, U0231, U0232, U0233, U0234, U0235, U0236, U0237, U0238, U0239, U023a, U023b, U023c, U023d, U023e, U023f, U0240, U0241, U0242, U0243, U0244, U0245, U0246, U0247, U0248, U0249, U024a, U024b, U024c, U024d, U024e, U024f, U0250, U0251, U0252, U0253, U0254, U0255, U0256, U0257, U0258, U0259, U025a, U025b, U025c, U025d, U025e, U025f, U0260, U0261, U0262, U0263, U0264, U0265, U0266, U0267, U0268, U0269, U026a, U026b, U026c, U026d, U026e, U026f, U0270, U0271, U0272, U0273, U0274, U0275, U0276, U0277, U0278, U0279, U027a, U027b, U027c, U027d, U027e, U027f, U0280, U0281, U0282, U0283, U0284, U0285, U0286, U0287, U0288, U0289, U028a, U028b, U028c, U028d, U028e, U028f, U0290, U0291, U0292, U0293, U0294, U0295, U0296, U0297, U0298, U0299, U029a, U029b, U029c, U029d, U029e, U029f, U02a0, U02a1, U02a2, U02a3, U02a4, U02a5, U02a6, U02a7, U02a8, U02a9, U02aa, U02ab, U02ac, U02ad, U02ae, U02af, U02b0, U02b1, U02b2, U02b3, U02b4, U02b5, U02b6, U02b7, U02b8, U02b9, U02ba, U02bb, U02bc, U02bd, U02be, U02bf, U02c0, U02c1, U02c2, U02c3, U02c4, U02c5, U02c6, U02c7, U02c8, U02c9, U02ca, U02cb, U02cc, U02cd, U02ce, U02cf, U02d0, U02d1, U02d2, U02d3, U02d4, U02d5, U02d6, U02d7, U02d8, U02d9, U02da, U02db, U02dc, U02dd, U02de, U02df, U02e0, U02e1, U02e2, U02e3, U02e4, U02e5, U02e6, U02e7, U02e8, U02e9, U02ea, U02eb, U02ec, U02ed, U02ee, U02ef, U02f0, U02f1, U02f2, U02f3, U02f4, U02f5, U02f6, U02f7, U02f8, U02f9, U02fa, U02fb, U02fc, U02fd, U02fe, U02ff, U0300, U0301, U0302, U0303, U0304, U0305, U0306, U0307, U0308, U0309, U030a, U030b, U030c, U030d, U030e, U030f, U0310, U0311, U0312, U0313, U0314, U0315, U0316, U0317, U0318, U0319, U031a, U031b, U031c, U031d, U031e, U031f, U0320, U0321, U0322, U0323, U0324, U0325, U0326, U0327, U0328, U0329, U032a, U032b, U032c, U032d, U032e, U032f, U0330, U0331, U0332, U0333, U0334, U0335, U0336, U0337, U0338, U0339, U033a, U033b, U033c, U033d, U033e, U033f, U0340, U0341, U0342, U0343, U0344, U0345, U0346, U0347, U0348, U0349, U034a, U034b, U034c, U034d, U034e, U034f, U0350, U0351, U0352, U0353, U0354, U0355, U0356, U0357, U0358, U0359, U035a, U035b, U035c, U035d, U035e, U035f, U0360, U0361, U0362, U0363, U0364, U0365, U0366, U0367, U0368, U0369, U036a, U036b, U036c, U036d, U036e, U036f, U0370, U0371, U0372, U0373, U0374, U0375, U0376, U0377, U037a, U037b, U037c, U037d, U037e, U0384, U0385, U0386, U0387, U0388, U0389, U038a, U038C, U038e, U038f, U0390, U0391, U0392, U0393, U0394, U0395, U0396, U0397, U0398, U0399, U039a, U039b, U039c, U039d, U039e, U039f, U03a0, U03a1, U03a3, U03a4, U03a5, U03a6, U03a7, U03a8, U03a9, U03aa, U03ab, U03ac, U03ad, U03ae, U03af, U03b0, U03b1, U03b2, U03b3, U03b4, U03b5, U03b6, U03b7, U03b8, U03b9, U03ba, U03bb, U03bc, U03bd, U03be, U03bf, U03c0, U03c1, U03c2, U03c3, U03c4, U03c5, U03c6, U03c7, U03c8, U03c9, U03ca, U03cb, U03cc, U03cd, U03ce, U03cf, U03d0, U03d1, U03d2, U03d3, U03d4, U03d5, U03d6, U03d7, U03d8, U03d9, U03da, U03db, U03dc, U03dd, U03de, U03df, U03e0, U03e1,

U03e2, U03e3, U03e4, U03e5, U03e6, U03e7, U03e8, U03e9, U03ea, U03eb, U03ec, U03ed, U03ee, U03ef, U03f0, U03f1, U03f2, U03f3, U03f4, U03f5, U03f6, U03f7, U03f8, U03f9, U03fa, U03fb, U03fc, U03fd, U03fe, U03ff, U0400, U0401, U0402, U0403, U0404, U0405, U0406, U0407, U0408, U0409, U040a, U040b, U040c, U040d, U040e, U040f, U0410, U0411, U0412, U0413, U0414, U0415, U0416, U0417, U0418, U0419, U041a, U041b, U041c, U041d, U041e, U041f, U0420, U0421, U0422, U0423, U0424, U0425, U0426, U0427, U0428, U0429, U042a, U042b, U042c, U042d, U042e, U042f, U0430, U0431, U0432, U0433, U0434, U0435, U0436, U0437, U0438, U0439, U043a, U043b, U043c, U043d, U043e, U043f, U0440, U0441, U0442, U0443, U0444, U0445, U0446, U0447, U0448, U0449, U044a, U044b, U044c, U044d, U044e, U044f, U0450, U0451, U0452, U0453, U0454, U0455, U0456, U0457, U0458, U0459, U045a, U045b, U045c, U045d, U045e, U045f, U0460, U0461, U0462, U0463, U0464, U0465, U0466, U0467, U0468, U0469, U046a, U046b, U046c, U046d, U046e, U046f, U0470, U0471, U0472, U0473, U0474, U0475, U0476, U0477, U0478, U0479, U047a, U047b, U047c, U047d, U047e, U047f, U0480, U0481, U0482, U0483, U0484, U0485, U0486, U0487, U0488, U0489, U048a, U048b, U048c, U048d, U048e, U048f, U0490, U0491, U0492, U0493, U0494, U0495, U0496, U0497, U0498, U0499, U049a, U049b, U049c, U049d, U049e, U049f, U04a0, U04a1, U04a2, U04a3, U04a4, U04a5, U04a6, U04a7, U04a8, U04a9, U04aa, U04ab, U04ac, U04ad, U04ae, U04af, U04b0, U04b1, U04b2, U04b3, U04b4, U04b5, U04b6, U04b7, U04b8, U04b9, U04ba, U04bb, U04bc, U04bd, U04be, U04bf, U04c0, U04c1, U04c2, U04c3, U04c4, U04c5, U04c6, U04c7, U04c8, U04c9, U04ca, U04cb, U04cc, U04cd, U04ce, U04cf, U04d0, U04d1, U04d2, U04d3, U04d4, U04d5, U04d6, U04d7, U04d8, U04d9, U04da, U04db, U04dc, U04dd, U04de, U04df, U04e0, U04e1, U04e2, U04e3, U04e4, U04e5, U04e6, U04e7, U04e8, U04e9, U04ea, U04eb, U04ec, U04ed, U04ee, U04ef, U04f0, U04f1, U04f2, U04f3, U04f4, U04f5, U04f6, U04f7, U04f8, U04f9, U04fa, U04fb, U04fc, U04fd, U04fe, U04ff, U0500, U0501, U0502, U0503, U0504, U0505, U0506, U0507, U0508, U0509, U050a, U050b, U050c, U050d, U050e, U050f, U0510, U0511, U0512, U0513, U0514, U0515, U0516, U0517, U0518, U0519, U051a, U051b, U051c, U051d, U051e, U051f, U0520, U0521, U0522, U0523, U0524, U0525, U0526, U0527, U0531, U0532, U0533, U0534, U0535, U0536, U0537, U0538, U0539, U053a, U053b, U053c, U053d, U053e, U053f, U0540, U0541, U0542, U0543, U0544, U0545, U0546, U0547, U0548, U0549, U054a, U054b, U054c, U054d, U054e, U054f, U0550, U0551, U0552, U0553, U0554, U0555, U0556, U0559, U055a, U055b, U055c, U055d, U055e, U055f, U0561, U0562, U0563, U0564, U0565, U0566, U0567, U0568, U0569, U056a, U056b, U056c, U056d, U056e, U056f, U0570, U0571, U0572, U0573, U0574, U0575, U0576, U0577, U0578, U0579, U057a, U057b, U057c, U057d, U057e, U057f, U0580, U0581, U0582, U0583, U0584, U0585, U0586, U0587, U0589, U058a, U058F, U0591, U0592, U0593, U0594, U0595, U0596, U0597, U0598, U0599, U059a, U059b, U059c, U059d, U059e, U059f, U05a0, U05a1, U05a2, U05a3, U05a4, U05a5, U05a6, U05a7, U05a8, U05a9, U05aa, U05ab, U05ac, U05ad, U05ae, U05af, U05b0, U05b1, U05b2, U05b3, U05b4, U05b5, U05b6, U05b7, U05b8, U05b9, U05ba, U05bb, U05bc, U05bd, U05be, U05bf, U05c0, U05c1, U05c2, U05c3, U05c4, U05c5, U05c6, U05c7, U05d0, U05d1, U05d2, U05d3, U05d4, U05d5, U05d6, U05d7, U05d8, U05d9, U05da, U05db, U05dc, U05dd, U05de, U05df, U05e0, U05e1,

U05e2, U05e3, U05e4, U05e5, U05e6, U05e7, U05e8, U05e9, U05ea, U05f0, U05f1, U05f2, U05f3, U05f4, U0600, U0601, U0602, U0603, U0606, U0607, U0608, U0609, U060a, U060b, U060c, U060d, U060e, U060f, U0610, U0611, U0612, U0613, U0614, U0615, U0616, U0617, U0618, U0619, U061a, U061b, U061e, U061f, U0620, U0621, U0622, U0623, U0624, U0625, U0626, U0627, U0628, U0629, U062a, U062b, U062c, U062d, U062e, U062f, U0630, U0631, U0632, U0633, U0634, U0635, U0636, U0637, U0638, U0639, U063a, U063b, U063c, U063d, U063e, U063f, U0640, U0641, U0642, U0643, U0644, U0645, U0646, U0647, U0648, U0649, U064a, U064b, U064c, U064d, U064e, U064f, U0650, U0651, U0652, U0653, U0654, U0655, U0656, U0657, U0658, U0659, U065a, U065b, U065c, U065d, U065e, U065f, U0660, U0661, U0662, U0663, U0664, U0665, U0666, U0667, U0668, U0669, U066a, U066b, U066c, U066d, U066e, U066f, U0670, U0671, U0672, U0673, U0674, U0675, U0676, U0677, U0678, U0679, U067a, U067b, U067c, U067d, U067e, U067f, U0680, U0681, U0682, U0683, U0684, U0685, U0686, U0687, U0688, U0689, U068a, U068b, U068c, U068d, U068e, U068f, U0690, U0691, U0692, U0693, U0694, U0695, U0696, U0697, U0698, U0699, U069a, U069b, U069c, U069d, U069e, U069f, U06a0, U06a1, U06a2, U06a3, U06a4, U06a5, U06a6, U06a7, U06a8, U06a9, U06aa, U06ab, U06ac, U06ad, U06ae, U06af, U06b0, U06b1, U06b2, U06b3, U06b4, U06b5, U06b6, U06b7, U06b8, U06b9, U06ba, U06bb, U06bc, U06bd, U06be, U06bf, U06c0, U06c1, U06c2, U06c3, U06c4, U06c5, U06c6, U06c7, U06c8, U06c9, U06ca, U06cb, U06cc, U06cd, U06ce, U06cf, U06d0, U06d1, U06d2, U06d3, U06d4, U06d5, U06d6, U06d7, U06d8, U06d9, U06da, U06db, U06dc, U06dd, U06de, U06df, U06e0, U06e1, U06e2, U06e3, U06e4, U06e5, U06e6, U06e7, U06e8, U06e9, U06ea, U06eb, U06ec, U06ed, U06ee, U06ef, U06f0, U06f1, U06f2, U06f3, U06f4, U06f5, U06f6, U06f7, U06f8, U06f9, U06fa, U06fb, U06fc, U06fd, U06fe, U06ff, U0750, U0751, U0752, U0753, U0754, U0755, U0756, U0757, U0758, U0759, U075a, U075b, U075c, U075d, U075e, U075f, U0760, U0761, U0762, U0763, U0764, U0765, U0766, U0767, U0768, U0769, U076a, U076b, U076c, U076d, U076e, U076f, U0770, U0771, U0772, U0773, U0774, U0775, U0776, U0777, U0778, U0779, U077a, U077b, U077c, U077d, U077e, U077f, U0e01, U0e02, U0e03, U0e04, U0e05, U0e06, U0e07, U0e08, U0e09, U0e0a, U0e0b, U0e0c, U0e0d, U0e0e, U0e0f, U0e10, U0e11, U0e12, U0e13, U0e14, U0e15, U0e16, U0e17, U0e18, U0e19, U0e1a, U0e1b, U0e1c, U0e1d, U0e1e, U0e1f, U0e20, U0e21, U0e22, U0e23, U0e24, U0e25, U0e26, U0e27, U0e28, U0e29, U0e2a, U0e2b, U0e2c, U0e2d, U0e2e, U0e2f, U0e30, U0e31, U0e32, U0e33, U0e34, U0e35, U0e36, U0e37, U0e38, U0e39, U0e3a, U0e3f, U0e40, U0e41, U0e42, U0e43, U0e44, U0e45, U0e46, U0e47, U0e48, U0e49, U0e4a, U0e4b, U0e4c, U0e4d, U0e4e, U0e4f, U0e50, U0e51, U0e52, U0e53, U0e54, U0e55, U0e56, U0e57, U0e58, U0e59, U0e5a, U0e5b, U1d00, U1d01, U1d02, U1d03, U1d04, U1d05, U1d06, U1d07, U1d08, U1d09, U1d0a, U1d0b, U1d0c, U1d0d, U1d0e, U1d0f, U1d10, U1d11, U1d12, U1d13, U1d14, U1d15, U1d16, U1d17, U1d18, U1d19, U1d1a, U1d1b, U1d1c, U1d1d, U1d1e, U1d1f, U1d20, U1d21, U1d22, U1d23, U1d24, U1d25, U1d26, U1d27, U1d28, U1d29, U1d2a, U1d2b, U1d2c, U1d2d, U1d2e, U1d2f, U1d30, U1d31, U1d32, U1d33, U1d34, U1d35, U1d36, U1d37, U1d38, U1d39, U1d3a, U1d3b, U1d3c, U1d3d, U1d3e, U1d3f, U1d40, U1d400, U1d401, U1d402, U1d403, U1d404, U1d405, U1d406, U1d407, U1d408, U1d409, U1d40a, U1d40b, U1d40c, U1d40d, U1d40e, U1d40f, U1d41, U1d410, U1d411, U1d412, U1d413, U1d414, U1d415, U1d416,

U1d417, U1d418, U1d419, U1d41a, U1d41b, U1d41c, U1d41d, U1d41e, U1d41f, U1d42, U1d420, U1d421, U1d422, U1d423, U1d424, U1d425, U1d426, U1d427, U1d428, U1d429, U1d42a, U1d42b, U1d42c, U1d42d, U1d42e, U1d42f, U1d43, U1d430, U1d431, U1d432, U1d433, U1d434, U1d435, U1d436, U1d437, U1d438, U1d439, U1d43a, U1d43b, U1d43c, U1d43d, U1d43e, U1d43f, U1d44, U1d440, U1d441, U1d442, U1d443, U1d444, U1d445, U1d446, U1d447, U1d448, U1d449, U1d44a, U1d44b, U1d44c, U1d44d, U1d44e, U1d44f, U1d45, U1d450, U1d451, U1d452, U1d453, U1d454, U1d456, U1d457, U1d458, U1d459, U1d45a, U1d45b, U1d45c, U1d45d, U1d45e, U1d45f, U1d46, U1d460, U1d461, U1d462, U1d463, U1d464, U1d465, U1d466, U1d467, U1d468, U1d469, U1d46a, U1d46b, U1d46c, U1d46d, U1d46e, U1d46f, U1d47, U1d470, U1d471, U1d472, U1d473, U1d474, U1d475, U1d476, U1d477, U1d478, U1d479, U1d47a, U1d47b, U1d47c, U1d47d, U1d47e, U1d47f, U1d48, U1d480, U1d481, U1d482, U1d483, U1d484, U1d485, U1d486, U1d487, U1d488, U1d489, U1d48a, U1d48b, U1d48c, U1d48d, U1d48e, U1d48f, U1d49, U1d490, U1d491, U1d492, U1d493, U1d494, U1d495, U1d496, U1d497, U1d498, U1d499, U1d49a, U1d49b, U1d49c, U1d49e, U1d49f, U1d4a, U1D4A2, U1d4a5, U1d4a6, U1d4a9, U1d4aa, U1d4ab, U1d4ac, U1d4ae, U1d4af, U1d4b, U1d4b0, U1d4b1, U1d4b2, U1d4b3, U1d4b4, U1d4b5, U1d4b6, U1d4b7, U1d4b8, U1d4b9, U1D4BB, U1d4bd, U1d4be, U1d4bf, U1d4c, U1d4c0, U1d4c1, U1d4c2, U1d4c3, U1d4c5, U1d4c6, U1d4c7, U1d4c8, U1d4c9, U1d4ca, U1d4cb, U1d4cc, U1d4cd, U1d4ce, U1d4cf, U1d4d, U1d4d0, U1d4d1, U1d4d2, U1d4d3, U1d4d4, U1d4d5, U1d4d6, U1d4d7, U1d4d8, U1d4d9, U1d4da, U1d4db, U1d4dc, U1d4dd, U1d4de, U1d4df, U1d4e, U1d4e0, U1d4e1, U1d4e2, U1d4e3, U1d4e4, U1d4e5, U1d4e6, U1d4e7, U1d4e8, U1d4e9, U1d4ea, U1d4eb, U1d4ec, U1d4ed, U1d4ee, U1d4ef, U1d4f, U1d4f0, U1d4f1, U1d4f2, U1d4f3, U1d4f4, U1d4f5, U1d4f6, U1d4f7, U1d4f8, U1d4f9, U1d4fa, U1d4fb, U1d4fc, U1d4fd, U1d4fe, U1d4ff, U1d50, U1d500, U1d501, U1d502, U1d503, U1d504, U1d505, U1d507, U1d508, U1d509, U1d50a, U1d50d, U1d50e, U1d50f, U1d51, U1d510, U1d511, U1d512, U1d513, U1d514, U1d516, U1d517, U1d518, U1d519, U1d51a, U1d51b, U1d51c, U1d51e, U1d51f, U1d52, U1d520, U1d521, U1d522, U1d523, U1d524, U1d525, U1d526, U1d527, U1d528, U1d529, U1d52a, U1d52b, U1d52c, U1d52d, U1d52e, U1d52f, U1d53, U1d530, U1d531, U1d532, U1d533, U1d534, U1d535, U1d536, U1d537, U1d538, U1d539, U1d53b, U1d53c, U1d53d, U1d53e, U1d54, U1d540, U1d541, U1d542, U1d543, U1d544, U1D546, U1d54a, U1d54b, U1d54c, U1d54d, U1d54e, U1d54f, U1d55, U1d550, U1d552, U1d553, U1d554, U1d555, U1d556, U1d557, U1d558, U1d559, U1d55a, U1d55b, U1d55c, U1d55d, U1d55e, U1d55f, U1d56, U1d560, U1d561, U1d562, U1d563, U1d564, U1d565, U1d566, U1d567, U1d568, U1d569, U1d56a, U1d56b, U1d56c, U1d56d, U1d56e, U1d56f, U1d57, U1d570, U1d571, U1d572, U1d573, U1d574, U1d575, U1d576, U1d577, U1d578, U1d579, U1d57a, U1d57b, U1d57c, U1d57d, U1d57e, U1d57f, U1d58, U1d580, U1d581, U1d582, U1d583, U1d584, U1d585, U1d586, U1d587, U1d588, U1d589, U1d58a, U1d58b, U1d58c, U1d58d, U1d58e, U1d58f, U1d59, U1d590, U1d591, U1d592, U1d593, U1d594, U1d595, U1d596, U1d597, U1d598, U1d599, U1d59a, U1d59b, U1d59c, U1d59d, U1d59e, U1d59f, U1d5a, U1d5a0, U1d5a1, U1d5a2, U1d5a3, U1d5a4, U1d5a5, U1d5a6, U1d5a7, U1d5a8, U1d5a9, U1d5aa, U1d5ab, U1d5ac, U1d5ad, U1d5ae, U1d5af, U1d5b, U1d5b0, U1d5b1, U1d5b2, U1d5b3, U1d5b4, U1d5b5, U1d5b6, U1d5b7, U1d5b8, U1d5b9, U1d5ba, U1d5bb, U1d5bc, U1d5bd, U1d5be, U1d5bf, U1d5c, U1d5c0, U1d5c1, U1d5c2, U1d5c3, U1d5c4, U1d5c5,

U1d5c6, U1d5c7, U1d5c8, U1d5c9, U1d5ca, U1d5cb, U1d5cc, U1d5cd, U1d5ce, U1d5cf, U1d5d, U1d5d0, U1d5d1, U1d5d2, U1d5d3, U1d5d4, U1d5d5, U1d5d6, U1d5d7, U1d5d8, U1d5d9, U1d5da, U1d5db, U1d5dc, U1d5dd, U1d5de, U1d5df, U1d5e, U1d5e0, U1d5e1, U1d5e2, U1d5e3, U1d5e4, U1d5e5, U1d5e6, U1d5e7, U1d5e8, U1d5e9, U1d5ea, U1d5eb, U1d5ec, U1d5ed, U1d5ee, U1d5ef, U1d5f, U1d5f0, U1d5f1, U1d5f2, U1d5f3, U1d5f4, U1d5f5, U1d5f6, U1d5f7, U1d5f8, U1d5f9, U1d5fa, U1d5fb, U1d5fc, U1d5fd, U1d5fe, U1d5ff, U1d60, U1d600, U1d601, U1d602, U1d603, U1d604, U1d605, U1d606, U1d607, U1d608, U1d609, U1d60a, U1d60b, U1d60c, U1d60d, U1d60e, U1d60f, U1d61, U1d610, U1d611, U1d612, U1d613, U1d614, U1d615, U1d616, U1d617, U1d618, U1d619, U1d61a, U1d61b, U1d61c, U1d61d, U1d61e, U1d61f, U1d62, U1d620, U1d621, U1d622, U1d623, U1d624, U1d625, U1d626, U1d627, U1d628, U1d629, U1d62a, U1d62b, U1d62c, U1d62d, U1d62e, U1d62f, U1d63, U1d630, U1d631, U1d632, U1d633, U1d634, U1d635, U1d636, U1d637, U1d638, U1d639, U1d63a, U1d63b, U1d63c, U1d63d, U1d63e, U1d63f, U1d64, U1d640, U1d641, U1d642, U1d643, U1d644, U1d645, U1d646, U1d647, U1d648, U1d649, U1d64a, U1d64b, U1d64c, U1d64d, U1d64e, U1d64f, U1d65, U1d650, U1d651, U1d652, U1d653, U1d654, U1d655, U1d656, U1d657, U1d658, U1d659, U1d65a, U1d65b, U1d65c, U1d65d, U1d65e, U1d65f, U1d66, U1d660, U1d661, U1d662, U1d663, U1d664, U1d665, U1d666, U1d667, U1d668, U1d669, U1d66a, U1d66b, U1d66c, U1d66d, U1d66e, U1d66f, U1d67, U1d670, U1d671, U1d672, U1d673, U1d674, U1d675, U1d676, U1d677, U1d678, U1d679, U1d67a, U1d67b, U1d67c, U1d67d, U1d67e, U1d67f, U1d68, U1d680, U1d681, U1d682, U1d683, U1d684, U1d685, U1d686, U1d687, U1d688, U1d689, U1d68a, U1d68b, U1d68c, U1d68d, U1d68e, U1d68f, U1d69, U1d690, U1d691, U1d692, U1d693, U1d694, U1d695, U1d696, U1d697, U1d698, U1d699, U1d69a, U1d69b, U1d69c, U1d69d, U1d69e, U1d69f, U1d6a, U1d6a0, U1d6a1, U1d6a2, U1d6a3, U1d6a4, U1d6a5, U1d6a8, U1d6a9, U1d6aa, U1d6ab, U1d6ac, U1d6ad, U1d6ae, U1d6af, U1d6b, U1d6b0, U1d6b1, U1d6b2, U1d6b3, U1d6b4, U1d6b5, U1d6b6, U1d6b7, U1d6b8, U1d6b9, U1d6ba, U1d6bb, U1d6bc, U1d6bd, U1d6be, U1d6bf, U1d6c, U1d6c0, U1d6c1, U1d6c2, U1d6c3, U1d6c4, U1d6c5, U1d6c6, U1d6c7, U1d6c8, U1d6c9, U1d6ca, U1d6cb, U1d6cc, U1d6cd, U1d6ce, U1d6cf, U1d6d, U1d6d0, U1d6d1, U1d6d2, U1d6d3, U1d6d4, U1d6d5, U1d6d6, U1d6d7, U1d6d8, U1d6d9, U1d6da, U1d6db, U1d6dc, U1d6dd, U1d6de, U1d6df, U1d6e, U1d6e0, U1d6e1, U1d6e2, U1d6e3, U1d6e4, U1d6e5, U1d6e6, U1d6e7, U1d6e8, U1d6e9, U1d6ea, U1d6eb, U1d6ec, U1d6ed, U1d6ee, U1d6ef, U1d6f, U1d6f0, U1d6f1, U1d6f2, U1d6f3, U1d6f4, U1d6f5, U1d6f6, U1d6f7, U1d6f8, U1d6f9, U1d6fa, U1d6fb, U1d6fc, U1d6fd, U1d6fe, U1d6ff, U1d70, U1d700, U1d701, U1d702, U1d703, U1d704, U1d705, U1d706, U1d707, U1d708, U1d709, U1d70a, U1d70b, U1d70c, U1d70d, U1d70e, U1d70f, U1d71, U1d710, U1d711, U1d712, U1d713, U1d714, U1d715, U1d716, U1d717, U1d718, U1d719, U1d71a, U1d71b, U1d71c, U1d71d, U1d71e, U1d71f, U1d72, U1d720, U1d721, U1d722, U1d723, U1d724, U1d725, U1d726, U1d727, U1d728, U1d729, U1d72a, U1d72b, U1d72c, U1d72d, U1d72e, U1d72f, U1d73, U1d730, U1d731, U1d732, U1d733, U1d734, U1d735, U1d736, U1d737, U1d738, U1d739, U1d73a, U1d73b, U1d73c, U1d73d, U1d73e, U1d73f, U1d74, U1d740, U1d741, U1d742, U1d743, U1d744, U1d745, U1d746, U1d747, U1d748, U1d749, U1d74a, U1d74b, U1d74c, U1d74d, U1d74e, U1d74f, U1d75, U1d750, U1d751, U1d752, U1d753, U1d754, U1d755, U1d756, U1d757, U1d758, U1d759, U1d75a, U1d75b, U1d75c, U1d75d, U1d75e, U1d75f,

U1d76, U1d760, U1d761, U1d762, U1d763, U1d764, U1d765, U1d766, U1d767, U1d768, U1d769, U1d76a, U1d76b, U1d76c, U1d76d, U1d76e, U1d76f, U1d77, U1d770, U1d771, U1d772, U1d773, U1d774, U1d775, U1d776, U1d777, U1d778, U1d779, U1d77a, U1d77b, U1d77c, U1d77d, U1d77e, U1d77f, U1d78, U1d780, U1d781, U1d782, U1d783, U1d784, U1d785, U1d786, U1d787, U1d788, U1d789, U1d78a, U1d78b, U1d78c, U1d78d, U1d78e, U1d78f, U1d79, U1d790, U1d791, U1d792, U1d793, U1d794, U1d795, U1d796, U1d797, U1d798, U1d799, U1d79a, U1d79b, U1d79c, U1d79d, U1d79e, U1d79f, U1d7a, U1d7a0, U1d7a1, U1d7a2, U1d7a3, U1d7a4, U1d7a5, U1d7a6, U1d7a7, U1d7a8, U1d7a9, U1d7aa, U1d7ab, U1d7ac, U1d7ad, U1d7ae, U1d7af, U1d7b, U1d7b0, U1d7b1, U1d7b2, U1d7b3, U1d7b4, U1d7b5, U1d7b6, U1d7b7, U1d7b8, U1d7b9, U1d7ba, U1d7bb, U1d7bc, U1d7bd, U1d7be, U1d7bf, U1d7c, U1d7c0, U1d7c1, U1d7c2, U1d7c3, U1d7c4, U1d7c5, U1d7c6, U1d7c7, U1d7c8, U1d7c9, U1d7ce, U1d7cf, U1d7d, U1d7d0, U1d7d1, U1d7d2, U1d7d3, U1d7d4, U1d7d5, U1d7d6, U1d7d7, U1d7d8, U1d7d9, U1d7da, U1d7db, U1d7dc, U1d7dd, U1d7de, U1d7df, U1d7e, U1d7e0, U1d7e1, U1d7e2, U1d7e3, U1d7e4, U1d7e5, U1d7e6, U1d7e7, U1d7e8, U1d7e9, U1d7ea, U1d7eb, U1d7ec, U1d7ed, U1d7ee, U1d7ef, U1d7f, U1d7f0, U1d7f1, U1d7f2, U1d7f3, U1d7f4, U1d7f5, U1d7f6, U1d7f7, U1d7f8, U1d7f9, U1d7fa, U1d7fb, U1d7fc, U1d7fd, U1d7fe, U1d7ff, U1d80, U1d81, U1d82, U1d83, U1d84, U1d85, U1d86, U1d87, U1d88, U1d89, U1d8a, U1d8b, U1d8c, U1d8d, U1d8e, U1d8f, U1d90, U1d91, U1d92, U1d93, U1d94, U1d95, U1d96, U1d97, U1d98, U1d99, U1d9a, U1d9b, U1d9c, U1d9d, U1d9e, U1d9f, U1da0, U1da1, U1da2, U1da3, U1da4, U1da5, U1da6, U1da7, U1da8, U1da9, U1daa, U1dab, U1dac, U1dad, U1dae, U1daf, U1db0, U1db1, U1db2, U1db3, U1db4, U1db5, U1db6, U1db7, U1db8, U1db9, U1dba, U1dbb, U1dbc, U1dbd, U1dbe, U1dbf, U1dc0, U1dc1, U1DC3, U1DCA, U1dfe, U1dff, U1e00, U1e01, U1e02, U1e03, U1e04, U1e05, U1e06, U1e07, U1e08, U1e09, U1e0a, U1e0b, U1e0c, U1e0d, U1e0e, U1e0f, U1e10, U1e11, U1e12, U1e13, U1e14, U1e15, U1e16, U1e17, U1e18, U1e19, U1e1a, U1e1b, U1e1c, U1e1d, U1e1e, U1e1f, U1e20, U1e21, U1e22, U1e23, U1e24, U1e25, U1e26, U1e27, U1e28, U1e29, U1e2a, U1e2b, U1e2c, U1e2d, U1e2e, U1e2f, U1e30, U1e31, U1e32, U1e33, U1e34, U1e35, U1e36, U1e37, U1e38, U1e39, U1e3a, U1e3b, U1e3c, U1e3d, U1e3e, U1e3f, U1e40, U1e41, U1e42, U1e43, U1e44, U1e45, U1e46, U1e47, U1e48, U1e49, U1e4a, U1e4b, U1e4c, U1e4d, U1e4e, U1e4f, U1e50, U1e51, U1e52, U1e53, U1e54, U1e55, U1e56, U1e57, U1e58, U1e59, U1e5a, U1e5b, U1e5c, U1e5d, U1e5e, U1e5f, U1e60, U1e61, U1e62, U1e63, U1e64, U1e65, U1e66, U1e67, U1e68, U1e69, U1e6a, U1e6b, U1e6c, U1e6d, U1e6e, U1e6f, U1e70, U1e71, U1e72, U1e73, U1e74, U1e75, U1e76, U1e77, U1e78, U1e79, U1e7a, U1e7b, U1e7c, U1e7d, U1e7e, U1e7f, U1e80, U1e81, U1e82, U1e83, U1e84, U1e85, U1e86, U1e87, U1e88, U1e89, U1e8a, U1e8b, U1e8c, U1e8d, U1e8e, U1e8f, U1e90, U1e91, U1e92, U1e93, U1e94, U1e95, U1e96, U1e97, U1e98, U1e99, U1e9a, U1e9b, U1e9c, U1e9d, U1e9e, U1e9f, U1ea0, U1ea1, U1ea2, U1ea3, U1ea4, U1ea5, U1ea6, U1ea7, U1ea8, U1ea9, U1eaa, U1eab, U1eac, U1ead, U1eae, U1eaf, U1eb0, U1eb1, U1eb2, U1eb3, U1eb4, U1eb5, U1eb6, U1eb7, U1eb8, U1eb9, U1eba, U1ebb, U1ebc, U1ebd, U1ebe, U1ebf, U1ec0, U1ec1, U1ec2, U1ec3, U1ec4, U1ec5, U1ec6, U1ec7, U1ec8, U1ec9, U1eca, U1ecb, U1ecc, U1ecd, U1ece, U1ecf, U1ed0, U1ed1, U1ed2, U1ed3, U1ed4, U1ed5, U1ed6, U1ed7, U1ed8, U1ed9, U1eda, U1edb, U1edc, U1edd, U1ede, U1edf, U1ee0, U1ee1, U1ee2, U1ee3, U1ee4, U1ee5, U1ee6, U1ee7, U1ee8, U1ee9, U1eea, U1eeb, U1eec, U1eed, U1eee, U1eef, U1ef0, U1ef1,

U1ef2, U1ef3, U1ef4, U1ef5, U1ef6, U1ef7, U1ef8, U1ef9, U1efa, U1efb, U1efc, U1efd, U1efe, U1eff, U1f00, U1f01, U1f02, U1f03, U1f04, U1f05, U1f06, U1f07, U1f08, U1f09, U1f0a, U1f0b, U1f0c, U1f0d, U1f0e, U1f0f, U1f10, U1f11, U1f12, U1f13, U1f14, U1f15, U1f18, U1f19, U1f1a, U1f1b, U1f1c, U1f1d, U1f20, U1f21, U1f22, U1f23, U1f24, U1f25, U1f26, U1f27, U1f28, U1f29, U1f2a, U1f2b, U1f2c, U1f2d, U1f2e, U1f2f, U1f30, U1f31, U1f32, U1f33, U1f34, U1f35, U1f36, U1f37, U1f38, U1f39, U1f3a, U1f3b, U1f3c, U1f3d, U1f3e, U1f3f, U1f40, U1f41, U1f42, U1f43, U1f44, U1f45, U1f48, U1f49, U1f4a, U1f4b, U1f4c, U1f4d, U1f50, U1f51, U1f52, U1f53, U1f54, U1f55, U1f56, U1f57, U1F59, U1F5B, U1F5D, U1f5f, U1f60, U1f61, U1f62, U1f63, U1f64, U1f65, U1f66, U1f67, U1f68, U1f69, U1f6a, U1f6b, U1f6c, U1f6d, U1f6e, U1f6f, U1f70, U1f71, U1f72, U1f73, U1f74, U1f75, U1f76, U1f77, U1f78, U1f79, U1f7a, U1f7b, U1f7c, U1f7d, U1f80, U1f81, U1f82, U1f83, U1f84, U1f85, U1f86, U1f87, U1f88, U1f89, U1f8a, U1f8b, U1f8c, U1f8d, U1f8e, U1f8f, U1f90, U1f91, U1f92, U1f93, U1f94, U1f95, U1f96, U1f97, U1f98, U1f99, U1f9a, U1f9b, U1f9c, U1f9d, U1f9e, U1f9f, U1fa0, U1fa1, U1fa2, U1fa3, U1fa4, U1fa5, U1fa6, U1fa7, U1fa8, U1fa9, U1faa, U1fab, U1fac, U1fad, U1fae, U1faf, U1fb0, U1fb1, U1fb2, U1fb3, U1fb4, U1fb6, U1fb7, U1fb8, U1fb9, U1fba, U1fbb, U1fbc, U1fbd, U1fbe, U1fbf, U1fc0, U1fc1, U1fc2, U1fc3, U1fc4, U1fc6, U1fc7, U1fc8, U1fc9, U1fca, U1fcb, U1fcc, U1fcd, U1fce, U1fcf, U1fd0, U1fd1, U1fd2, U1fd3, U1fd6, U1fd7, U1fd8, U1fd9, U1fda, U1fdb, U1fdd, U1fde, U1fdf, U1fe0, U1fe1, U1fe2, U1fe3, U1fe4, U1fe5, U1fe6, U1fe7, U1fe8, U1fe9, U1fea, U1feb, U1fec, U1fed, U1fee, U1fef, U1ff2, U1ff3, U1ff4, U1ff6, U1ff7, U1ff8, U1ff9, U1ffa, U1ffb, U1ffc, U1ffd, U1ffe, U2000, U2001, U2002, U2003, U2004, U2005, U2006, U2007, U2008, U2009, U200a, U200b, U200c, U200d, U200e, U200f, U2010, U2011, U2012, U2013, U2014, U2015, U2016, U2017, U2018, U2019, U201a, U201b, U201c, U201d, U201e, U201f, U2020, U2021, U2022, U2025, U2026, U2028, U2029, U202a, U202b, U202c, U202d, U202e, U202f, U2030, U2031, U2032, U2033, U2034, U2035, U2036, U2037, U2038, U2039, U203a, U203b, U203c, U203d, U203e, U2040, U2043, U2044, U2047, U204e, U204f, U2050, U2051, U2052, U2057, U205e, U205f, U206a, U206b, U206c, U206d, U206e, U206f, U2070, U2074, U2075, U2076, U2077, U2078, U2079, U207f, U2080, U2081, U2082, U2083, U2084, U2085, U2086, U2087, U2088, U2089, U2090, U2091, U2092, U2093, U2094, U20a0, U20a1, U20a2, U20a3, U20a4, U20a5, U20a6, U20a7, U20a8, U20a9, U20aa, U20ab, U20ac, U20ad, U20ae, U20af, U20b0, U20b1, U20b2, U20b3, U20b4, U20b5, U20b6, U20b7, U20b8, U20b9, U20ba, U20d0, U20d1, U20d2, U20d6, U20d7, U20db, U20dc, U20dd, U20de, U20df, U20E1, U20e4, U20e5, U20e6, U20e7, U20e8, U20e9, U20ea, U20eb, U20ec, U20ed, U20ee, U20ef, U20f0, U2102, U2105, U2107, U210a, U210b, U210c, U210d, U210e, U210f, U2110, U2111, U2112, U2113, U2115, U2116, U2117, U2118, U2119, U211a, U211b, U211c, U211d, U211e, U2120, U2122, U2124, U2125, U2126, U2127, U2128, U2129, U212b, U212c, U212d, U212e, U212f, U2130, U2131, U2132, U2133, U2134, U2135, U2136, U2137, U2138, U213c, U213d, U213e, U213f, U2140, U2141, U2142, U2143, U2144, U2145, U2146, U2147, U2148, U2149, U214a, U214b, U214d, U214e, U2153, U2154, U2155, U2156, U2157, U2158, U2159, U215a, U215b, U215c, U215d, U215e, U2184, U2190, U2191, U2192, U2193, U2194, U2195, U2196, U2197, U2198, U2199, U219a, U219b, U219c, U219d, U219e, U219f, U21a0, U21a1, U21a2, U21a3, U21a4, U21a5, U21a6, U21a7, U21a8, U21a9, U21aa, U21ab, U21ac, U21ad,

U21ae, U21af, U21b0, U21b1, U21b2, U21b3, U21b4, U21b5, U21b6, U21b7, U21b8, U21b9, U21ba, U21bb, U21bc, U21bd, U21be, U21bf, U21c0, U21c1, U21c2, U21c3, U21c4, U21c5, U21c6, U21c7, U21c8, U21c9, U21ca, U21cb, U21cc, U21cd, U21ce, U21cf, U21d0, U21d1, U21d2, U21d3, U21d4, U21d5, U21d6, U21d7, U21d8, U21d9, U21da, U21db, U21dc, U21dd, U21de, U21df, U21e0, U21e1, U21e2, U21e3, U21e4, U21e5, U21e6, U21e7, U21e8, U21e9, U21ea, U21f4, U21f5, U21f6, U21f7, U21f8, U21f9, U21fa, U21fb, U21fc, U21fd, U21fe, U21ff, U2200, U2201, U2202, U2203, U2204, U2205, U2206, U2207, U2208, U2209, U220a, U220b, U220c, U220d, U220e, U220f, U2210, U2211, U2212, U2213, U2214, U2215, U2216, U2217, U2218, U2219, U221a, U221b, U221c, U221d, U221e, U221f, U2220, U2221, U2222, U2223, U2224, U2225, U2226, U2227, U2228, U2229, U222a, U222b, U222c, U222d, U222e, U222f, U2230, U2231, U2232, U2233, U2234, U2235, U2236, U2237, U2238, U2239, U223a, U223b, U223c, U223d, U223e, U223f, U2240, U2241, U2242, U2243, U2244, U2245, U2246, U2247, U2248, U2249, U224a, U224b, U224c, U224d, U224e, U224f, U2250, U2251, U2252, U2253, U2254, U2255, U2256, U2257, U2258, U2259, U225a, U225b, U225c, U225d, U225e, U225f, U2260, U2261, U2262, U2263, U2264, U2265, U2266, U2267, U2268, U2269, U226a, U226b, U226c, U226d, U226e, U226f, U2270, U2271, U2272, U2273, U2274, U2275, U2276, U2277, U2278, U2279, U227a, U227b, U227c, U227d, U227e, U227f, U2280, U2281, U2282, U2283, U2284, U2285, U2286, U2287, U2288, U2289, U228a, U228b, U228c, U228d, U228e, U228f, U2290, U2291, U2292, U2293, U2294, U2295, U2296, U2297, U2298, U2299, U229a, U229b, U229c, U229d, U229e, U229f, U22a0, U22a1, U22a2, U22a3, U22a4, U22a5, U22a6, U22a7, U22a8, U22a9, U22aa, U22ab, U22ac, U22ad, U22ae, U22af, U22b0, U22b1, U22b2, U22b3, U22b4, U22b5, U22b6, U22b7, U22b8, U22b9, U22ba, U22bb, U22bc, U22bd, U22be, U22bf, U22c0, U22c1, U22c2, U22c3, U22c4, U22c5, U22c6, U22c7, U22c8, U22c9, U22ca, U22cb, U22cc, U22cd, U22ce, U22cf, U22d0, U22d1, U22d2, U22d3, U22d4, U22d5, U22d6, U22d7, U22d8, U22d9, U22da, U22db, U22dc, U22dd, U22de, U22df, U22e0, U22e1, U22e2, U22e3, U22e4, U22e5, U22e6, U22e7, U22e8, U22e9, U22ea, U22eb, U22ec, U22ed, U22ee, U22ef, U22f0, U22f1, U22f2, U22f3, U22f4, U22f5, U22f6, U22f7, U22f8, U22f9, U22fa, U22fb, U22fc, U22fd, U22fe, U22ff, U2300, U2302, U2305, U2306, U2308, U2309, U230a, U230b, U230c, U230d, U230e, U230f, U2310, U2311, U2312, U2313, U2315, U2316, U2317, U2318, U2319, U231a, U231c, U231d, U231e, U231f, U2320, U2321, U2322, U2323, U2329, U232a, U232c, U232d, U232e, U2332, U2336, U233D, U233f, U2340, U2353, U2370, U237C, U2393, U2394, U23AF, U23b4, U23b5, U23b6, U23CE, U23D0, U23dc, U23dd, U23de, U23df, U23e0, U23e1, U23e2, U23e3, U23e4, U23e5, U23e6, U23e7, U2423, U2460, U2461, U2462, U2463, U2464, U2465, U2466, U2467, U2468, U24b6, U24b7, U24b8, U24b9, U24ba, U24bb, U24bc, U24bd, U24be, U24bf, U24c0, U24c1, U24c2, U24c3, U24c4, U24c5, U24c6, U24c7, U24c8, U24c9, U24ca, U24cb, U24cc, U24cd, U24ce, U24cf, U24d0, U24d1, U24d2, U24d3, U24d4, U24d5, U24d6, U24d7, U24d8, U24d9, U24da, U24db, U24dc, U24dd, U24de, U24df, U24e0, U24e1, U24e2, U24e3, U24e4, U24e5, U24e6, U24e7, U24e8, U24e9, U24ea, U2500, U2502, U2506, U2508, U250A, U250C, U2510, U2514, U2518, U251C, U2524, U252C, U2534, U253C, U2550, U2551, U2552, U2553, U2554, U2555, U2556, U2557, U2558, U2559, U255a, U255b, U255c, U255d, U255e, U255f, U2560,

U2561, U2562, U2563, U2564, U2565, U2566, U2567, U2568, U2569, U256a, U256b, U256c, U2571, U2572, U2580, U2584, U2588, U258C, U2590, U2591, U2592, U2593, U25a0, U25a1, U25a2, U25a3, U25a4, U25a5, U25a6, U25a7, U25a8, U25a9, U25aa, U25ab, U25ac, U25ad, U25ae, U25af, U25b0, U25b1, U25b2, U25b3, U25b4, U25b5, U25b6, U25b7, U25b8, U25b9, U25ba, U25bb, U25bc, U25bd, U25be, U25bf, U25c0, U25c1, U25c2, U25c3, U25c4, U25c5, U25c6, U25c7, U25c8, U25c9, U25ca, U25cb, U25cc, U25cd, U25ce, U25cf, U25d0, U25d1, U25d2, U25d3, U25d4, U25d5, U25d6, U25d7, U25d8, U25d9, U25da, U25db, U25dc, U25dd, U25de, U25df, U25e0, U25e1, U25e2, U25e3, U25e4, U25e5, U25e6, U25e7, U25e8, U25e9, U25ea, U25eb, U25ec, U25ed, U25ee, U25ef, U25f0, U25f1, U25f2, U25f3, U25f4, U25f5, U25f6, U25f7, U25f8, U25f9, U25fa, U25fb, U25fc, U25fd, U25fe, U25ff, U2605, U2606, U2609, U260C, U260E, U2612, U2621, U2639, U263a, U263b, U263c, U263d, U263e, U263f, U2640, U2641, U2642, U2643, U2644, U2646, U2647, U2648, U2649, U2660, U2661, U2662, U2663, U2664, U2665, U2666, U2667, U2669, U266a, U266b, U266d, U266e, U266f, U267E, U2680, U2681, U2682, U2683, U2684, U2685, U2686, U2687, U2688, U2689, U26A0, U26A5, U26aa, U26ab, U26ac, U26B2, U26E2, U2702, U2709, U2713, U2720, U272A, U2736, U273D, U2772, U2773, U2780, U2781, U2782, U2783, U2784, U2785, U2786, U2787, U2788, U2789, U278a, U278b, U278c, U278d, U278e, U278f, U2790, U2791, U2792, U2793, U279B, U27c0, U27c1, U27c2, U27c3, U27c4, U27c5, U27c6, U27c7, U27c8, U27c9, U27cb, U27cc, U27cd, U27d0, U27d1, U27d2, U27d3, U27d4, U27d5, U27d6, U27d7, U27d8, U27d9, U27da, U27db, U27dc, U27dd, U27de, U27df, U27e0, U27e1, U27e2, U27e3, U27e4, U27e5, U27e6, U27e7, U27e8, U27e9, U27ea, U27eb, U27ec, U27ed, U27ee, U27ef, U27f0, U27f1, U27f2, U27f3, U27f4, U27f5, U27f6, U27f7, U27f8, U27f9, U27fa, U27fb, U27fc, U27fd, U27fe, U27ff, U2900, U2901, U2902, U2903, U2904, U2905, U2906, U2907, U2908, U2909, U290a, U290b, U290c, U290d, U290e, U290f, U2910, U2911, U2912, U2913, U2914, U2915, U2916, U2917, U2918, U2919, U291a, U291b, U291c, U291d, U291e, U291f, U2920, U2921, U2922, U2923, U2924, U2925, U2926, U2927, U2928, U2929, U292a, U292b, U292c, U292d, U292e, U292f, U2930, U2931, U2932, U2933, U2934, U2935, U2936, U2937, U2938, U2939, U293a, U293b, U293c, U293d, U293e, U293f, U2940, U2941, U2942, U2943, U2944, U2945, U2946, U2947, U2948, U2949, U294a, U294b, U294c, U294d, U294e, U294f, U2950, U2951, U2952, U2953, U2954, U2955, U2956, U2957, U2958, U2959, U295a, U295b, U295c, U295d, U295e, U295f, U2960, U2961, U2962, U2963, U2964, U2965, U2966, U2967, U2968, U2969, U296a, U296b, U296c, U296d, U296e, U296f, U2970, U2971, U2972, U2973, U2974, U2975, U2976, U2977, U2978, U2979, U297a, U297b, U297c, U297d, U297e, U297f, U2980, U2981, U2982, U2983, U2984, U2985, U2986, U2987, U2988, U2989, U298a, U298b, U298c, U298d, U298e, U298f, U2990, U2991, U2992, U2993, U2994, U2995, U2996, U2997, U2998, U2999, U299a, U299b, U299c, U299d, U299e, U299f, U29a0, U29a1, U29a2, U29a3, U29a4, U29a5, U29a6, U29a7, U29a8, U29a9, U29aa, U29ab, U29ac, U29ad, U29ae, U29af, U29b0, U29b1, U29b2, U29b3, U29b4, U29b5, U29b6, U29b7, U29b8, U29b9, U29ba, U29bb, U29bc, U29bd, U29be, U29bf, U29c0, U29c1, U29c2, U29c3, U29c4, U29c5, U29c6, U29c7, U29c8, U29c9, U29ca, U29cb, U29cc, U29cd, U29ce, U29cf, U29d0, U29d1, U29d2, U29d3, U29d4, U29d5, U29d6, U29d7, U29d8, U29d9, U29da, U29db, U29dc, U29dd, U29de,

U29df, U29e0, U29e1, U29e2, U29e3, U29e4, U29e5, U29e6, U29e7, U29e8, U29e9, U29ea, U29eb, U29ec, U29ed, U29ee, U29ef, U29f0, U29f1, U29f2, U29f3, U29f4, U29f5, U29f6, U29f7, U29f8, U29f9, U29fa, U29fb, U29fc, U29fd, U29fe, U29ff, U2a00, U2a01, U2a02, U2a03, U2a04, U2a05, U2a06, U2a07, U2a08, U2a09, U2a0a, U2a0b, U2a0c, U2a0d, U2a0e, U2a0f, U2a10, U2a11, U2a12, U2a13, U2a14, U2a15, U2a16, U2a17, U2a18, U2a19, U2a1a, U2a1b, U2a1c, U2a1d, U2a1e, U2a1f, U2a20, U2a21, U2a22, U2a23, U2a24, U2a25, U2a26, U2a27, U2a28, U2a29, U2a2a, U2a2b, U2a2c, U2a2d, U2a2e, U2a2f, U2a30, U2a31, U2a32, U2a33, U2a34, U2a35, U2a36, U2a37, U2a38, U2a39, U2a3a, U2a3b, U2a3c, U2a3d, U2a3e, U2a3f, U2a40, U2a41, U2a42, U2a43, U2a44, U2a45, U2a46, U2a47, U2a48, U2a49, U2a4a, U2a4b, U2a4c, U2a4d, U2a4e, U2a4f, U2a50, U2a51, U2a52, U2a53, U2a54, U2a55, U2a56, U2a57, U2a58, U2a59, U2a5a, U2a5b, U2a5c, U2a5d, U2a5e, U2a5f, U2a60, U2a61, U2a62, U2a63, U2a64, U2a65, U2a66, U2a67, U2a68, U2a69, U2a6a, U2a6b, U2a6c, U2a6d, U2a6e, U2a6f, U2a70, U2a71, U2a72, U2a73, U2a74, U2a75, U2a76, U2a77, U2a78, U2a79, U2a7a, U2a7b, U2a7c, U2a7d, U2a7e, U2a7f, U2a80, U2a81, U2a82, U2a83, U2a84, U2a85, U2a86, U2a87, U2a88, U2a89, U2a8a, U2a8b, U2a8c, U2a8d, U2a8e, U2a8f, U2a90, U2a91, U2a92, U2a93, U2a94, U2a95, U2a96, U2a97, U2a98, U2a99, U2a9a, U2a9b, U2a9c, U2a9d, U2a9e, U2a9f, U2aa0, U2aa1, U2aa2, U2aa3, U2aa4, U2aa5, U2aa6, U2aa7, U2aa8, U2aa9, U2aaa, U2aab, U2aac, U2aad, U2aae, U2aaf, U2ab0, U2ab1, U2ab2, U2ab3, U2ab4, U2ab5, U2ab6, U2ab7, U2ab8, U2ab9, U2aba, U2abb, U2abc, U2abd, U2abe, U2abf, U2ac0, U2ac1, U2ac2, U2ac3, U2ac4, U2ac5, U2ac6, U2ac7, U2ac8, U2ac9, U2aca, U2acb, U2acc, U2acd, U2ace, U2acf, U2ad0, U2ad1, U2ad2, U2ad3, U2ad4, U2ad5, U2ad6, U2ad7, U2ad8, U2ad9, U2ada, U2adb, U2adc, U2add, U2ade, U2adf, U2ae0, U2ae1, U2ae2, U2ae3, U2ae4, U2ae5, U2ae6, U2ae7, U2ae8, U2ae9, U2aea, U2aeb, U2aec, U2aed, U2aee, U2aef, U2af0, U2af1, U2af2, U2af3, U2af4, U2af5, U2af6, U2af7, U2af8, U2af9, U2afa, U2afb, U2afc, U2afd, U2afe, U2aff, U2b12, U2b13, U2b14, U2b15, U2b16, U2b17, U2b18, U2b19, U2b1a, U2b1b, U2b1c, U2b1d, U2b1e, U2b1f, U2b20, U2b21, U2b22, U2b23, U2b24, U2b25, U2b26, U2b27, U2b28, U2b29, U2b2a, U2b2b, U2b2c, U2b2d, U2b2e, U2b2f, U2b30, U2b31, U2b32, U2b33, U2b34, U2b35, U2b36, U2b37, U2b38, U2b39, U2b3a, U2b3b, U2b3c, U2b3d, U2b3e, U2b3f, U2b40, U2b41, U2b42, U2b43, U2b44, U2b45, U2b46, U2b47, U2b48, U2b49, U2b4a, U2b4b, U2b4c, U2b50, U2b51, U2b52, U2b53, U2b54, U2c60, U2c61, U2c62, U2c63, U2c64, U2c65, U2c66, U2c67, U2c68, U2c69, U2c6a, U2c6b, U2c6c, U2c6d, U2c6e, U2c6f, U2c70, U2c71, U2c72, U2c73, U2c74, U2c75, U2c76, U2c77, U2c78, U2c79, U2c7a, U2c7b, U2c7c, U2c7d, U2c7e, U2c7f, U2E17, U3012, U3030, U306E, Ua717, Ua718, Ua719, Ua71a, Ua71b, Ua71c, Ua71d, Ua71e, Ua71f, Ua720, Ua721, UA727, Ua788, Ua789, Ua78a, Ua78b, Ua78c, UA792, UE000, Ufb00, Ufb01, Ufb02, Ufb03, Ufb04, Ufb13, Ufb14, Ufb15, Ufb16, Ufb17, Ufb1d, Ufb1e, Ufb1f, Ufb20, Ufb21, Ufb22, Ufb23, Ufb24, Ufb25, Ufb26, Ufb27, Ufb28, Ufb29, Ufb2a, Ufb2b, Ufb2c, Ufb2d, Ufb2e, Ufb2f, Ufb30, Ufb31, Ufb32, Ufb33, Ufb34, Ufb35, Ufb36, Ufb38, Ufb39, Ufb3a, Ufb3b, Ufb3c, UFB3E, Ufb40, Ufb41, Ufb43, Ufb44, Ufb46, Ufb47, Ufb48, Ufb49, Ufb4a, Ufb4b, Ufb4c, Ufb4d, Ufb4e, Ufb4f, Ufb50, Ufb51, Ufb52, Ufb53, Ufb54, Ufb55, Ufb56, Ufb57, Ufb58, Ufb59, Ufb5a, Ufb5b, Ufb5c, Ufb5d, Ufb5e, Ufb5f, Ufb60, Ufb61, Ufb62, Ufb63, Ufb64, Ufb65, Ufb66, Ufb67, Ufb68, Ufb69, Ufb6a, Ufb6b, Ufb6c, Ufb6d, Ufb6e, Ufb6f,

Ufb70, Ufb71, Ufb72, Ufb73, Ufb74, Ufb75, Ufb76, Ufb77, Ufb78, Ufb79, Ufb7a, Ufb7b, Ufb7c, Ufb7d, Ufb7e, Ufb7f, Ufb80, Ufb81, Ufb82, Ufb83, Ufb84, Ufb85, Ufb86, Ufb87, Ufb88, Ufb89, Ufb8a, Ufb8b, Ufb8c, Ufb8d, Ufb8e, Ufb8f, Ufb90, Ufb91, Ufb92, Ufb93, Ufb94, Ufb95, Ufb96, Ufb97, Ufb98, Ufb99, Ufb9a, Ufb9b, Ufb9c, Ufb9d, Ufb9e, Ufb9f, Ufba0, Ufba1, Ufba2, Ufba3, Ufba4, Ufba5, Ufba6, Ufba7, Ufba8, Ufba9, Ufbaa, Ufbab, Ufbac, Ufbad, Ufbae, Ufbaf, Ufbb0, Ufbb1, Ufbb2, Ufbb3, Ufbb4, Ufbb5, Ufbb6, Ufbb7, Ufbb8, Ufbb9, Ufbba, Ufbbb, Ufbbc, Ufbbd, Ufbbe, Ufbbf, Ufbc0, Ufbc1, Ufbd3, Ufbd4, Ufbd5, Ufbd6, Ufbd7, Ufbd8, Ufbd9, Ufbda, Ufbdb, Ufbdc, Ufbdd, Ufbde, Ufbdf, Ufbe0, Ufbe1, Ufbe2, Ufbe3, Ufbe4, Ufbe5, Ufbe6, Ufbe7, Ufbe8, Ufbe9, Ufbfc, Ufbfd, Ufbfe, Ufbff, Ufc5e, Ufc5f, Ufc60, Ufc61, Ufc62, Ufc63, UFC6A, UFC6D, UFC70, UFC73, UFC91, UFC94, Ufd3e, Ufd3f, UFDF2, UFDFC, Ufe20, Ufe21, Ufe22, Ufe23, Ufe70, Ufe71, Ufe72, Ufe73, Ufe74, Ufe76, Ufe77, Ufe78, Ufe79, Ufe7a, Ufe7b, Ufe7c, Ufe7d, Ufe7e, Ufe7f, Ufe80, Ufe81, Ufe82, Ufe83, Ufe84, Ufe85, Ufe86, Ufe87, Ufe88, Ufe89, Ufe8a, Ufe8b, Ufe8c, Ufe8d, Ufe8e, Ufe8f, Ufe90, Ufe91, Ufe92, Ufe93, Ufe94, Ufe95, Ufe96, Ufe97, Ufe98, Ufe99, Ufe9a, Ufe9b, Ufe9c, Ufe9d, Ufe9e, Ufe9f, Ufea0, Ufea1, Ufea2, Ufea3, Ufea4, Ufea5, Ufea6, Ufea7, Ufea8, Ufea9, Ufeaa, Ufeab, Ufeac, Ufead, Ufeae, Ufeaf, Ufeb0, Ufeb1, Ufeb2, Ufeb3, Ufeb4, Ufeb5, Ufeb6, Ufeb7, Ufeb8, Ufeb9, Ufeba, Ufebb, Ufebc, Ufebd, Ufebe, Ufebf, Ufec0, Ufec1, Ufec2, Ufec3, Ufec4, Ufec5, Ufec6, Ufec7, Ufec8, Ufec9, Ufeca, Ufecb, Ufecc, Ufecd, Ufece, Ufecf, Ufed0, Ufed1, Ufed2, Ufed3, Ufed4, Ufed5, Ufed6, Ufed7, Ufed8, Ufed9, Ufeda, Ufedb, Ufedc, Ufedd, Ufede, Ufedf, Ufee0, Ufee1, Ufee2, Ufee3, Ufee4, Ufee5, Ufee6, Ufee7, Ufee8, Ufee9, Ufeea, Ufeeb, Ufeec, Ufeed, Ufeee, Ufeef, Ufef0, Ufef1, Ufef2, Ufef3, Ufef4, Ufef5, Ufef6, Ufef7, Ufef8, Ufef9, Ufefa, Ufefb, Ufefc, Ufffc, Ufffd