City University London
MSc Data Science (part-time) Project Report
2019

How can recent advances in Convolutional Neural Networks and Deep Learning improve the Optical Character Recognition of Persian text?

Marc Kendal
Supervised by Chris Child and Sepehr Jalili.
January 2019

Abstract

The aim of this project is to use deep learning algorithms to improve Optical character recognition (OCR) of the Persian alphabet. Recent advances in deep learning using Convolutional Neural Networks (CNNs) are used to classify Persian letters, numbers and words. Additionally, new approaches are investigated in image segmentation when there is semantic complexity. Residual Neural Networks are used to achieve 99.7% classification accuracy.

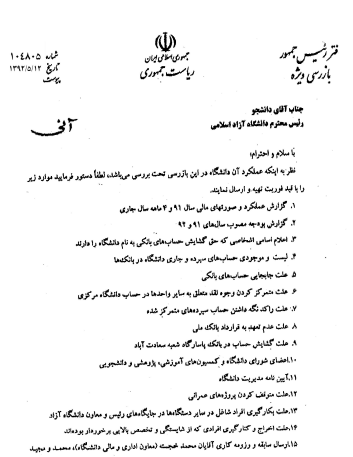
1 Introduction and Objectives

The overall aim of this research is to use deep learning algorithms to improve Optical character recognition (OCR) of the Persian alphabet. OCR can help Persian businesses use document image processing to recognise the contents of documents. More specifically, research is needed to accurately extract Persian numbers and letters and digitise them such that the information contained in the documents can be searched by key words. Currently, if a business that speaks and writes in Persian wants to know the contents of its internal documents, a worker would need to manually input the information in each document one by one. Sepehr Jalali et al, 2018 has created a data set that can be used to help to solve this contemporary business problem.

1.1 Farsi Optical Character Recognition Accuracy

OCR is commonly used and is effective in English and other Latin languages and can be easily accomplished through scanning and photography. However, in Persian and similar languages with related alphabets like Arabic and Urdu, research is still ongoing to create the technology for a similar capability. OCR text correction for Farsi and similar language are still being developed as seen in [1]. Higher quality images help to increase accuracy but often books and documents are lower quality and often hand written. Additionally, contemporary OCR models employ auto correct as a final layer, before classification using; "Language models or recognition dictionaries" [2]. However, "using a language model complicates training of OCR systems, and it also narrows the range of texts that an OCR system can be used with." [2] "Performance on non-linguistic data like codified alphanumerical strings is significantly worse." [3]

Persian also known as Farsi, is one of the Western Iranian languages within the Indo-Iranian branch of the Indo-European language family. This contrasts with Arabic which is considered a Central Semitic language [4]. However, Persian uses a modified variant of the Arabic script and therefore its relevant to consider OCR research on Arabic alphabet. Here is a couple of examples of the Persian business letters which are going to be used as test images for the OCR solution being developed:



1.2 Image Segmentation

Research will be undertaken to identify the Persian words in documents using Image Segmentation. There are three main challenges to this. The Letter data set has a low level of resolution, which might affect the ability to classify the Letters. Additionally, each letter is structured differently such that the information that is needed from each letter exists in different parts of the page. It can't be anticipated where to direct the algorithm to look.

1.3 Literature Contribution

In this project, I hope to contribute to the body of knowledge on Farsi OCR.
Persian also known as Farsi, is one of the Western Iranian languages within the Indo-Iranian branch of the Indo-European language family. This is in contrast to Arabic which is considered a Central Semitic language [2]. However, Persian uses a modified variant of the Arabic script and therefore its relevant to consider OCR research on Arabic alphabet.

It is interesting to see across the literature a similar assertion of the challenge of OCR for non-Latin scripts and specifically arabic. "(Arabic) is a cursive script…morphologically very rich text…characters are often connected by a baseline thereby forming sub-words. Moreover, one Arabic character may have more than one shape depending on its position in the word. This text has also different texture characteristics compared to Latin or Chinese ones: more stokes in different directions, different font size and aspect ratio, etc. All these particularities make…most (of the) existing OCR systems fail to detect and recognize Arabic Text."[3]

As shown, there are challenges of having an accurate OCR for Arabic alphabet and by extension the Persian alphabet. In the existing literature, many difference approaches and tools have been used. I looked at OCR literature in general and more specifically at OCR done on Arabic, and Urdu scripts. The current OCR research on the Persian alphabet is very limited, so I was looking to see what insights could be gleaned from the general literature and non-Latin scripts more specifically.

Another important challenge that makes this project unique as that there are not many Farsi data sets that have been used to train convolutional neural networks in the literature. Therefore, it is anticipated that any image recognition that will occur, will be based on the learning from the Persian data set is being used for this project. It is will be interesting to see what results can be achieved given that the data set is relatively small and of a low resolution.

1.4 Plan of work

There are three main tasks for this project. The first is to be able to use deep learning to accurately recognise where the date is on the page. Then second, once the location is found, to recognise the numbers which represent the dates of the documents and extract this metadata. Once the recognition of numbers has been achieved then, additional tasks include, optical character recognition of letters and words, including the content of the letters. Further work could include the OCR of hand written Persian.

2 Context

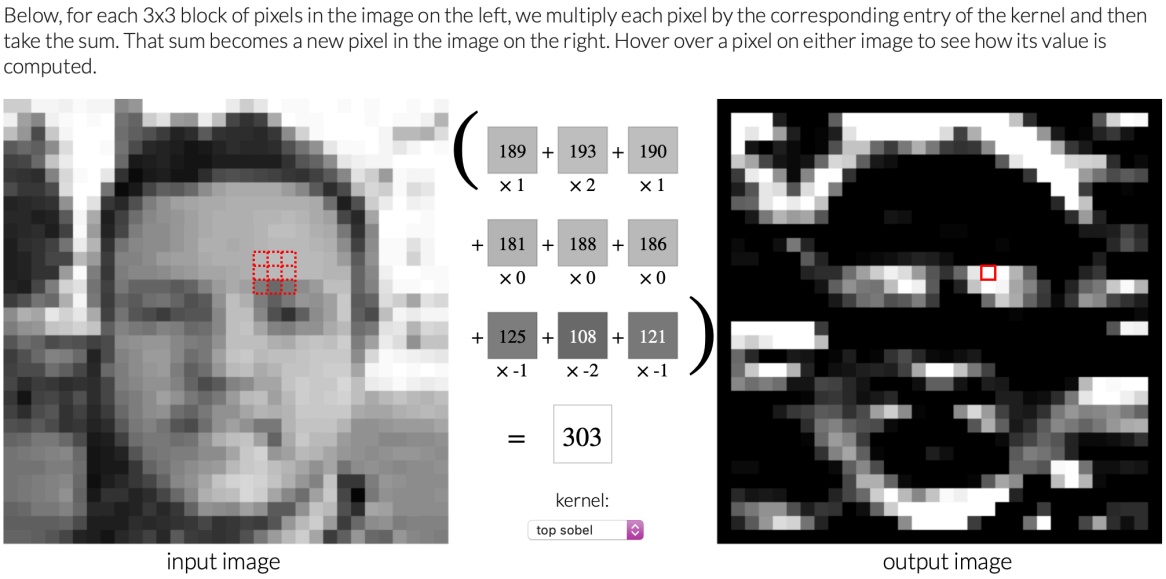
2.1 Infrastructure

When choosing the best infrastructure to run the algorithms I compared the different cloud server offerings. I wanted to get something workable, cheap and easy to use. Additionally, I found that using Pytorch together with FastAI codebase, since it is so new, made it hard to find infrastructure that loaded all of the dependencies needed and everything ran. The Pytorch[4] code base and the Fast AI wrapper[5] for PyTorch were chosen due to my interest in using cutting edge methods in deep learning.
I was in conversation with a Chief Digital Officer at a large media company who recalled how there were occasion when they left servers running on AWS and it cost them thousands of dollars per day. I investigated both AWS and Google Cloud and could see how it was easy to make a mistake and sign up to an expensive server, so although a GPU was needed, it needed to be affordable. As I was just learning how to run machine learning on the cloud it was important to me that I shouldn't get a large bill. Fortunately, Paperspace and Salamander were two options where the cost was still manageable, even if I did by mistake left the server on (something which I did a few times over the course of the project). Salamander seemed promising, because the environments has been configured for each version of the fastAI code base. However, after testing it out, it was shown too be too slow even on basic coding operations. Paperspace had two options, one to use a server and one to use a Jupyter notebook. I chose to use the server, and I was initially able to use it for the older FastAI code base version 0.7. However, I wanted to be able to use state of the art algorithms and their latest methods in version 1.0. I wasn't able to configure the server to have all the dependencies working together, including the numerous anaconda libraries, PyTorch, CUDA which put the model on the GPU[6] and the latest FastAI updates. However, Paperspace also offers a notebook facility that allows using a notebook where all the dependencies are already configured and crucially a notebook facility is available where the latest FastAI code build is working with all its associated dependencies, albeit in experimental BETA mode. However, unlike with a server, the jupyter notebook facility only allows the uploading of one file at a time. So, I created a Tar achive which compressed all the data into one file and I could upload the file through the jupyter interface and then unzip using the Tar library as shown in the Appendix.

2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN's) are a type of neural network that processes data that is displayed in a grid. For this project, image data can be arranged on a two-dimensional grid and therefore can be processed using a CNN. A CNN uses convolutions instead of regular matrix multiplication. A convolution is similar to a matrix multiplier but with useful properties for image classification. A convolutional is then described by the application of a convolutional operation to an input, resulting in an output. In the case of images, a kernel is applied to the input image and produces what is called a feature map.

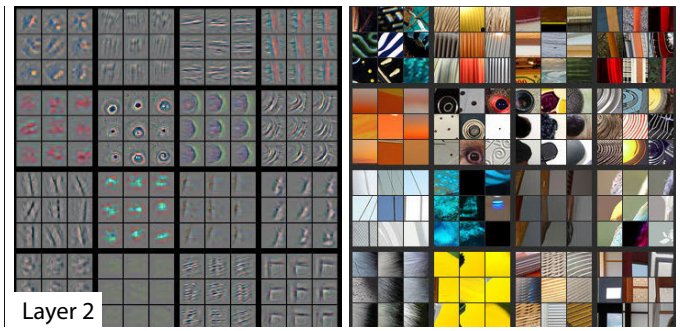
$$\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$



Source: http://setosa.io/ev/image-kernels/

For example, this website shows how a convolution works. As one moves around the picture with the three by three red square, the colours change and the colour values of the pixels change. In the centre, the values are shown for each pixel in the 3x3 block for each part of the picture. The pixel values are between 0 and 255 corresponding to the RGB value. Above the centre, there is a matrix which is a called a convolutional kernel. For each part of the image, Each of the nine pixel values in the 3x3 box is element-wise multiplied by each of the nine values in the convolutional kernel. The sum of these values is used to create a middle pixel in the image on the right.
There is one pixel less on the outside of the right image than the left, and it is replaced with a black border. This is due to the pixel value created being the located in the centre of the 3x3.
As can be seen the face has been turned into a picture where white represents the distinct horizontal edges. The intuition behind this can be explained as follows. When there is white shades in the top three boxes, (given that the RGB(red, green and blue) value of white is 255) they are getting multiplied by (1, 2, 1). However, the lower boxes are getting multiplied by (-1, -2, -1), but since the lower boxes are dark and therefore have values around zero, The total value of the convolution is going to be higher than 255 and therefore bright white. However, on the head where there is dark hair, the RGB values are low. Additionally, considering the colours as we go lower, the forehead is lighter, and therefore possess slightly higher RGB values. These values will be multiplied by negative values (-1, -2, -1) leading to a highly negative output in general which is represented by a dark forehead in the image on the right

Of note is a paper by Zeiler and Fergus that describe visually how convolutions work in layers.[7] Their paper describes how one can visualize the layers of a convolutional network. In convolutional neural networks the result of the first convolutional network becomes an input for the next convolution.



Source (Zeiler and Fergus 2014, Visualizing and understanding Convolutional Networks)

For example, this is the second layer. In the image on the left, each small box is a coefficient of the second layer, that operates on groups of pixels that are next to each other. The second layer has taken the results of the filters of the previous layer and does a second layer computation. It has learned to look for different characteristics in the image.


Source (Zeiler and Fergus 2014, Visualizing and understanding Convolutional Networks)

This is the bottom right 3x3 box. It shows how the CNN has learned to recognize top left corners.


Source (Zeiler and Fergus 2014, Visualizing and understanding Convolutional Networks)

One can see this by looking at the lower right 3x3 box displayed above which is also in the large image on the right. Each little box within this 3x3 box above, contains an instance of an image that the filter classified as containing a left-hand corner. As the layers increase, Zeiler and Fergus show visually how the filters are combined towards greater visual complexity.

The output of one convolution is called a channel. As the layers increase these channels are combined to create more detailed filters. In fact, as more feature complexity is required, CNNs use more kernals such that the output channels increase to the depth of the amount of kernals that are used. This is because each kernal function when it accepts an input produces its own output channel. As the network gets deeper, more channels are needed to capture increasingly complex features.

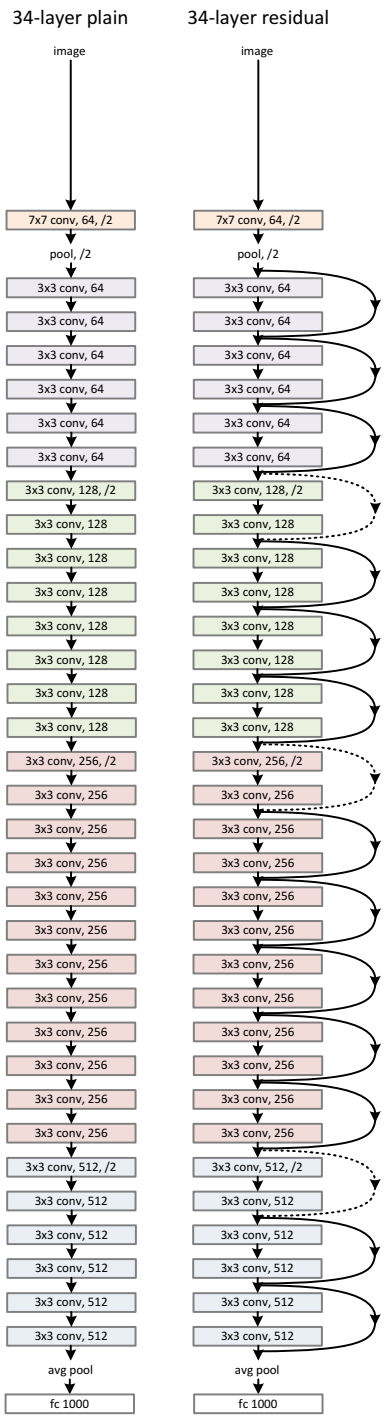
One can see an example of this in the following simplified summary of the learning layers of a CNN called Resnet 34, which was applied on the Persian Data set. For a more detailed summary of the network see the code in appendix.

Layer (type)	Output Shape	Param #	Trainable
Conv2d	[16, 64, 176, 176]	9408	False
Conv2d	[16, 64, 88, 88]	36864	False
Conv2d	[16, 128, 44, 44]	73728	False
Conv2d	[16, 256, 22, 22]	294912	False
Conv2d	[16, 512, 11, 11]	1179648	False
AdaptiveAvgPool2d	[16, 512, 1, 1]	0	False

Sometimes, to reduce memory consumption a stride convolution is used. This is where input pixels are skipped over and not evaluated. For example, a Stride convolution of two, skips every alternate input pixel and creates outputs channels that are half the height and width. This halving of the size allows for a doubling of the number of channels.

As can be seen, the output shape of the first layer, [16, 64, 176, 176], can be understood as follows: The size of the original input image is specified to be 352 x 352 pixels, and the first layer has a stride of 2 and, therefore the first layer has an output size of 176 x 176 pixels. The first layer has 64 output channels. This is because the grid size is halved, the number of channels is generally doubled.

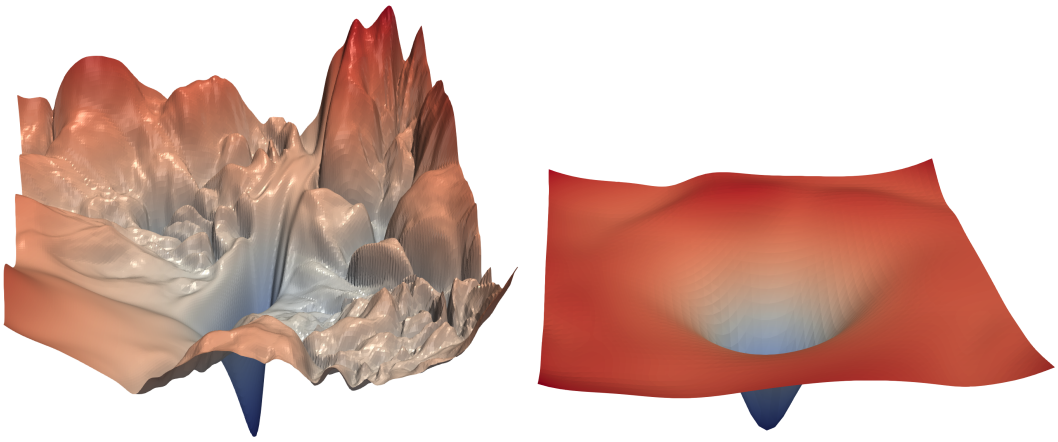
It is a residual network with 34 parameter layers [8]. Residual nets have shown greater accuracy than plain networks. In their paper, Deep Residual Learning for Image Recognition, He, Zhang, Ren and Sun describe how residual nets have greatly increased depth, through their architecture that inserts shortcut connections into a regular network. They state that, “The deep plain nets suffer from increased depth, and exhibit higher training error when going deeper,”and that they “provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth.” [8] See below an image of the Resnet architecture:



Comparison of network architectures for ImageNet for Resnet34
Left: a plain network with 34 parameter layers
Right: a residual network with 34 parameter layers. The dotted shortcuts increase dimensions. Source: Deep Residual Learning for Image Recognition by He, Zhang, Ren and Sun 2015

Convolutional blocks are stacked together to form a deep network[9]. One can compare a plain network with a residual network. In a residual network, connections are skipped to create 'short-cut' connections. This change in the architecture in connection of layers forms residual blocks as can been see in the residual network in the right image above. The key benefit of a Resnet is that it allows a deeper network without training loss. If a standard optimization algorithm, such as gradient descent is used to train a plain network, as the number of layers increases the training error will tend to decrease but after that it will increase.

In theory, a deeper network should decrease the training error but empirically the training error increases if the network is too deep. However, with Resnets, the training error can continue to decrease even when training a deep network, for example with hundreds of layers. Empirically, "ResNet can outperform all the state-of-the-art models on … small datasets."[10][11]



(a) without skip connections (b) with skip connections

These vivid images show the loss surfaces of a Resnet 56, with and without skip connections[12]. They provide a visual aid that helps to give an intuition as to why skip connections can be useful. Instead of a complex landscape like the loss surface in image (a), the skip connections create a 'smoother', less 'harsh landscape', shown in image (b) making it easier to arrive at a global minimum of loss.
I created an example of a convolution for two categories in the Persian data set. See below:

Original image Convolution Heat Map

The set of images in first row is the number ‘6’ in Persian. The second row is the word ‘Number’ in Persian.

To produce the convolution images, a matrix was used that focused on a finding a feature of bottom right edges, like the example shown in the Zeiler and Fergus paper. The matrix used is the following:

[0. ,-5/3,1],
[-5/3,-5/3,1],
[1. ,1 ,1]

The full code can be found in the appendix. One can see how the distribution of values in the matrix echoes the concept of both a right and bottom edge, as the values on the right and bottom are equal to one.
By understanding how to create a heat map, an intuition can be gained how the convolutional neural network can use the image channels and kernels to classify images. One can start by examining the final convolution layer where there are 512 image channels and the grid size is 11 x 11. This has been shown in the summary of convolutional layers above. In this final convolutional layer, the average value of each of the images is calculated using a process called average pooling.

By running the code “learn.model” in pytorch, one obtains the description of the convolutional neural network. In this description, the following function is shown which the does an average pooling operation:

(ap): AdaptiveAvgPool2d(output_size=1)

This average pooling operation in Pytorch is ‘2d’ because it takes an input that is composed of “several input planes” and takes an average value of each plane[13]. It returns a vector of width that is specified by the output size (in this case = 1) and height that is the number of channels or ‘planes’, which in our case is

512. To use this vector to make predictions, one can multiply it by a single matrix which is in the CNN represented by an additional linear layer that the CNN concludes with. This linear layer has an input of 512 and output of 18:

(8): Linear(in_features=512, out_features=18, bias=True)

There are 18 out features corresponding to the features in our data set. However, there are also what are called 'in features' which are the 512 image channels. During what is termed the 'forward pass' in Pytorch[14], the CNN has identified 512 output channels which can also be thought of as features and will be explained below. This Linear function reduces and converts the 512 output channels which can be thought of as features to the 18 features that are in our data set. This conversion to 18 features will need to represent the information learned from all 512 channels. More specifically, for the result of this Linear matrix computation to exhibit a low accuracy loss, it will need to represent a simple weighted linear combination of all the 512 input values. For this to be possible, these 512 input values will need to represent features such as how many corner edges and at what angles, and are there any curves, do they include these letters or not etc. The weights represent, for each of the 512 'in features', how much of this feature is represented in each data category 1 to 18. By thinking about the 11x11 grid before the average pooling function, AdaptiveAvgPool2d(output_size=1), one can consider how each 11 x 11 face represents one of 512 different features and how each pixel value within each 11x11, conveys to what extent this pixel is like each feature, for example, a corner at 90 degrees or a curve from bottom left to top right.

With this understanding one can create a heat map. An image of '6' and an image of 'number' was 'forward' passed into the CNN, after saving ('hooking') the output of the convolutional section of the model, which had shape of 512 x 11 x 11. Instead of averaging across the 11 x 11 grid as before, one can average across the 512 output channels, which will create a single 11 x 11 matrix. Each grid value in the matrix then represents how activated this area was, when considering the classification of a single data category. As examples, the images above contained the heat maps for the number '6' and the word 'number' in Persian.

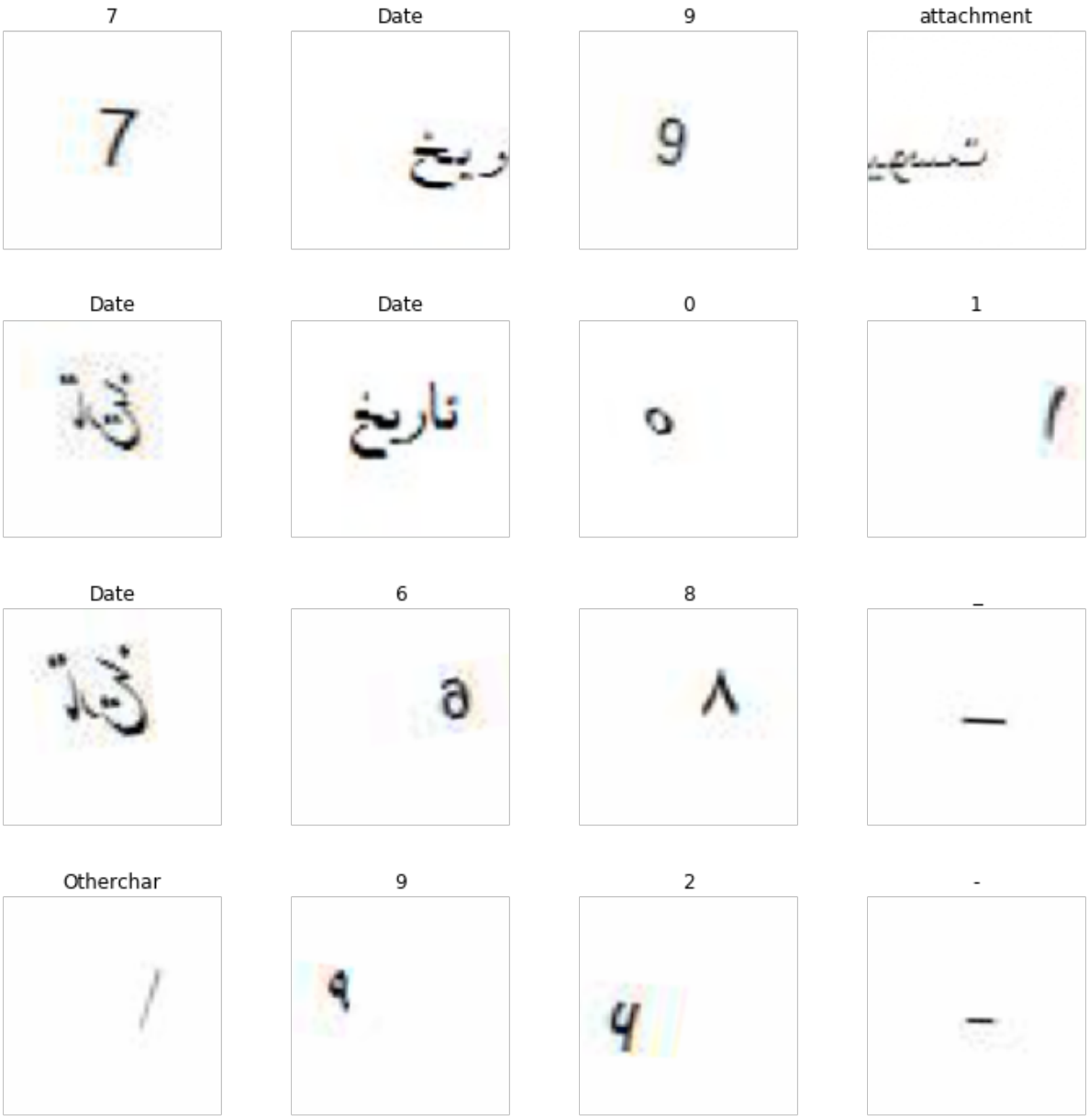
This was possible by using a PyTorch function called 'Hook' which allows "inspecting / modifying the output … of a layer" [15] In this case, it allows the 'hooking' of the output channels, before the average pool calculations are completed. Additionally, I utilized the 'hook_output' function from Fast AI to make use of this hook of the convolutional layer[16]. Matplotlib was used to display the image.

3 Methods

3.1 Data Set

The data set includes business documents written in Persian. These are of different types and topics such as employment documents and business transactions. On each document, there is metadata at the top of the page on the left. There are 3 fields that can be filled in; 'Title' which will be filled in with the title, 'dated' which will be filled in with the date and 'Attachment' which will be filled in if the document is an attachment.

The data set contains 1156 labelled images for all the numbers, 0 – 9, in Persian. Additionally, there are 1156 labelled images for the Persian words for 'Date', 'dated', 'attachment', 'number' and 'title'. The data set has been created such that each image showing the label is in a slightly different font. Then there are 59 images of Persian documents.



Here are some examples of the images of the classes in the data set.

To have the data in the format that ImageNet datasets are usually structured in, I wrote a python scripts to first transform the file names to include their class names, for example for the first image in the class 'Date' was named Date.01.jpg
After that the data structure could be transformed from a structure of class1….class18 to:

train\ class1\ class2\
valid\ class1\ class2\
test\

In hindsight, once the filenames had been renamed, the function ImageDataBunch.from_name_re could have been used. Instead I wrote a bespoke python function, see appendix.

3.2 Server

When I was using a server rather than a notebook instance I wrote code to upload it to the server. (See Appendix.) I also learned about using the ssh function in bash for remote logging in.

3.3 Image Batches and Normalisation

I created a python notebook to investigate if the data could be classified correctly.
The ImageDataBunch class [17] was used to take in the entire data set including the validation, training and test sets as well as the labels. It creates an image object that could be passed into the CNN algorithm and use the appropriate aspect of the data set when needed. The data set was already split into, training, validation and testing subsets, in the style of an Imagenet data set, which is a common way to arrange image data. Therefore, it was straightforward to use the from_folder factory method as the data had been pre-sorted via a custom python script as mentioned above. It was necessary to specify the size of the images even though the images already have a size. A GPU applies the identical instructions which at present requires all the images to be the same size. I started by setting the size to be 224 x 224 which is a common size. As can be seen above I printed out a random subset of the images, and the names of the labels to examine for any obvious errors.

The ImageDataBunch class also has a parameter for something called get_transforms which crops and resizes the images. There is also an option for data augmentation within the get_transforms function, however it is not recommended when dealing with text images because the meaning of text depends on the angle that it is being read. By turning a letter upside down the meaning could totally change. However, by turning a picture of a dog upside down for example, doesn't affect its meaning. An upside down dog is still classified as a dog.

Additionally, it has been shown empirically that normalizing data increasing performance of deep learning. In this case the images exist in three colour channels, red, green and blue. Each channel was normalised to have a mean of zero and standard deviation of one.

3.4 Learn CNN methods

I utilized a function called create_cnn to create a convolutional neural network. It uses the ImageDataBunch object that was created and a parameter that specifies the architecture. I started with a Resnet of size 34 instead of a larger size, so that it would train faster, which is useful when testing out the network. Afterwards I ran it on a deeper Resnet, referred to as Resnet 50. The function create_cnn also has a parameter for specifying which metrics one can print out and the error rate metric was chosen in order to understand how accurate the performance of the algorithm was.

Pytorch provides downloadable Resnet models that are pretrained. This means that they contain weights that were pre trained on around one and a half million pictures of different subjects and labels that are in the ImageNet dataset[18]. To make it work for the Persian Data set, Transfer Learning is used and will be described below in section 3.4. The goal of transfer learning is to fit the CNN to the ones data set. For

example, instead of being able to classify the one thousand ImageNet categories, it will classify the 18 classes of Persian letters, numbers, and words accurately. The benefit of using transfer learning is that it drastically reduces the time and the size of the data set needed to achieve the same accuracy.

However, when using transfer learning on a small data set and generally when training a model, it is important to make sure that overfitting doesn't occur, ie. that the training on the data set leads to learning, not memorisation, and consequently classification is generalizable rather than specific.[19]. There are several potential methods to reduce overfitting, including gathering more data, data augmentation[10], model simplification, early termination of training, regularization and dropout.[20] By keeping the validation set of images separate, and not shown to the model during its training, I was able to test for overfitting by comparing the error rate metrics of the training set and validation set.

I used the fit_one_cycle function, which utilises recent advances in model fitting[21] to vary the learning rate during the training of the model. In 'Fit one cycle' the learning rate starts low, increase and then decreases. If the learning rate is too high, the model may diverge completely or be very slow to converge. If the learning rate is too small to start off with, it can get stuck in local minima but not in a global minimum. The problem with finding a local minimum is that the model tends not to generalise well. However, by gradually increasing the learning rate, the model gets closer to the underlying ground truth quickly and the training loss is more likely to arrive at its global minimum. The increase in the learning rate, stops the model becoming stuck at minimum with a higher loss. Instead, it can 'break out' of the trough it is in and, explore the loss surface to arrive at a flatter area of the loss curve that has a lower loss value.

3.5 Fine tuning

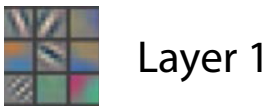
3.5.1 Transfer Learning, Freezing and Unfreezing.

In section 2.2, it was described how a CNN contains a multi-layer architecture. These multiple layers allow for combining channels to create more detailed filters. When these layers are trained, they in a certain sense, contain knowledge and possess filters which can classify the features which were learned. Before training on the new dataset, the model is pre-trained and therefore already contains feature weights that it learned from ImageNet dataset that the model is trained on (see section 3.4).

Instead of training the model from scratch, transfer learning allows a 'transfer' of the knowledge that was previously learned. It does this by training extra convolutional layers. These extra layers which are added to the end of the pre-trained model. Initially a Resnet 34 was used.

The first step was using the fit_one_cycle method as described above in section 3.3 to fit the model on the new data. The pre-trained layers were 'frozen'. This means that the fit_one_cycle method, only trains the new layers and doesn't back propagate the weights into the pre-trained layers. 'Freezing' the pre-trained layers and just training on some extra layers is beneficial as it is fast to train and the potential for overfitting is low. However, to improve performance, one can fine-tune the model, including the rest of the layers. To do this, the 'frozen' layers are 'unfrozen' An unfreeze method was used on the 'learn' object to accomplish this.
It has been suggested that gradual unfreezing produces better accuracy than just unfreezing all the layers, "as this contains the least general knowledge (Yosinski et al., 2014) ..(They) first unfreeze the last layer and fine-tune all unfrozen layers for one epoch." [22]. However, one can also reason that it depends on the data. If the new data set is similar to the pre-trained data set in its complexity or features, one could argue that it is isn't necessary to unfreeze the earlier layers. This is because the earlier pre-trained layers usually contain more simple and general filters.

One can gain an intuition of this concept, be examining images in the paper by Zeiler and Fergus mentioned in section 2.2.

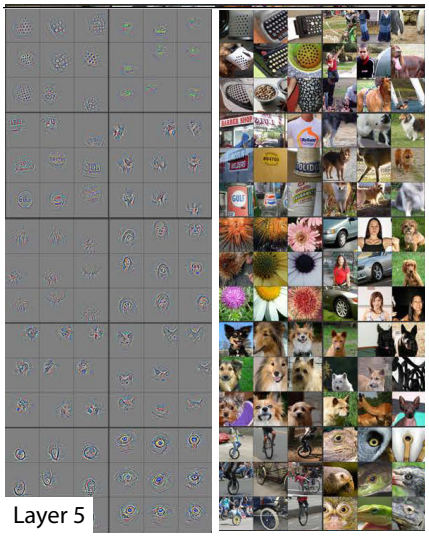


Source (Zeiler and Fergus 2014, Visualizing and understanding Convolutional Networks)

In the earlier layers the features created by the convolution are more simple and less recognizable as specific objects. Instead, the filters are more simple and general. For example, one can see in layer one, groups of pixels that look like diagonal lines in different directions and, simple colour gradients, blue to yellow and red to green. The results of these filters are also relatively not complex.

Source (Zeiler and Fergus 2014, Visualizing and understanding Convolutional Networks)

However by the fifth layer, already the complexity can be seen below:

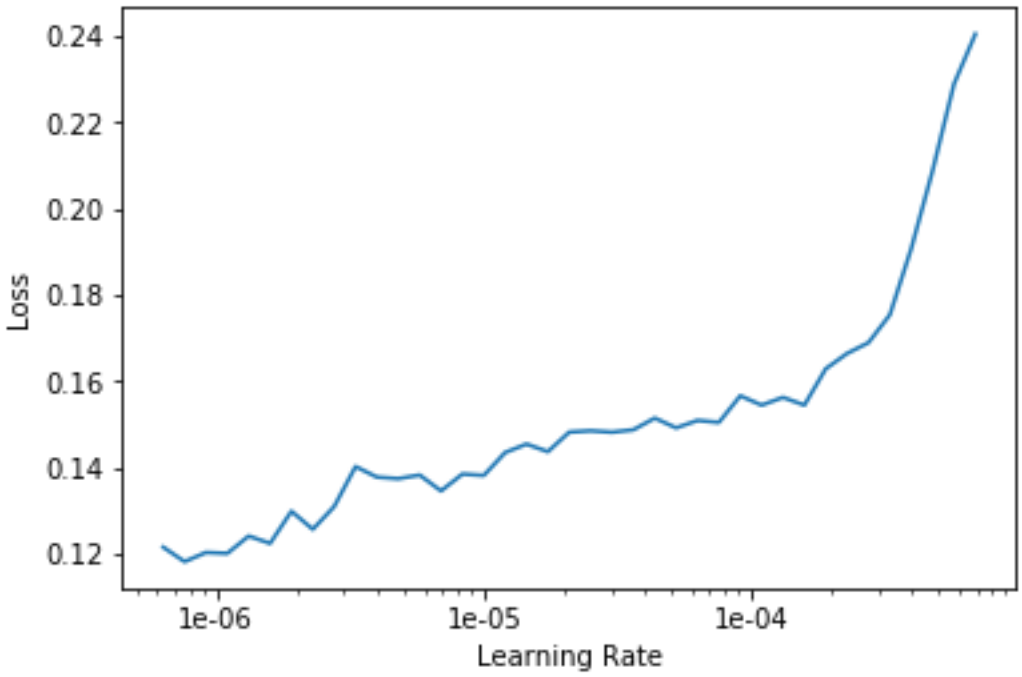


Source (Zeiler and Fergus 2014, Visualizing and understanding Convolutional Networks)

The filters in layer five recognize different types of objects and animals. As the layers increase the semantic complexity increases.
Initially the pre-trained layers were frozen by default and the last layers were trained for four epochs. I ran this code a few different times from the beginning to see if there was a consistent result that would be achieved. A range of accuracy was obtained, between 98% and 99.2% depending on the code versions used and the which attempt it was.

3.5.2 Learning rate

Afterwards the model was unfrozen, the model was trained for five more epochs. The error rate achieved was 0.26 %, in other words an accuracy of 99.73%, which is very impressive. I investigated to see whether the specifying the learning rate more specifically would have to produce better accuracy. There are many papers written on this approach[21], [23]–[26].
The learning rate finder method was used to find the maximum learning rate that could be used to train the neural network that would increase and not decrease the accuracy already gained. Since the model was saved after its initial training, it was possible to test how the model responded to being trained at different learning rate ranges.
To work out the best range of learning rates that the model should be fine-tuned on, the following graph was produced.



The graph plots the loss value that occurs as the learning rate varies. The loss is calculated as the exponentially weighted moving average of the losses, which helps to make the loss graph smoother and easier to read. As the learning rate moves towards 1e-4 which equals 0.0001, the loss starts to significantly increase. The default learning rate is 0.003. It can be reasoned from the graph; this it will be helpful to train at a lower rate than the default to reduce the loss value. On the other hand, it's important to not have the rate to be too low as it can get stuck at local minima. The fit_one_cycle method takes in a learning rate parameter via the slice function. A start value and stop value specifies the range of learning rates to use. "The first group's learning rate is start, the last is end, and the remaining are evenly geometrically spaced"[27]. Discriminatory learning rates are used, whereby the earlier layers are given a lower learning rate than the later layers.

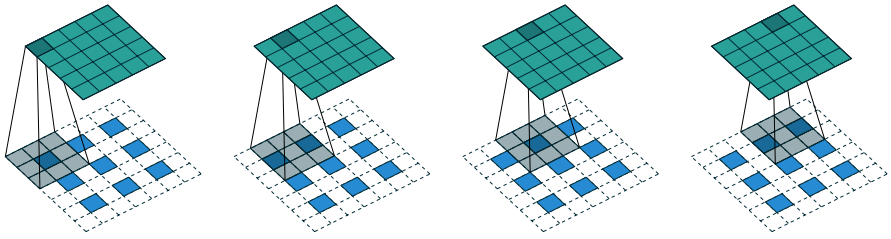
The expectation is that the earlier layers need less training due to the relatively less semantic complexity in the filters of the earlier layers, for example, diagonal lines versus more specific picture filters in the later layers. I tried out different learning rate ranges to see if the accuracy could be improved on the 99.73% achieved when no learning rate range was specified. By looking at the plot of the learning rate and loss, I specified the following ranges. (1e-6,1e-4), (1e-6,1e-3), (1e-5,1e-3), (1e-5,1e-4). After five epochs, the

same error rate of 0.32% accuracy was, for each test range. This error is marginally more than the 0.26% achieved before and therefore there wasn't any benefit shown to the fine-tuning method of specifying the learning rate. See appendix for Jupyter Notebook. One could suggest that since the complexity of the Persian data set is semantically more simple than the ImageNet data set, a higher learning rate was needed to retrain some of the earlier layers which might more share characterizes to images of text. Further work could investigate which types of image data perform better when fine tuning the learning rate.

Also, there are different opinions about whether if the validation loss is higher than the training loss, that this is a sign of overfitting. In this case of the training of the model on the Persian data, it was usually the training loss which was bigger than the validation loss. Therefore, a case could be made for training the model for more epochs. If so, it would be important to save the model in stages and track back to the previous stage if overfitting was occurring. It would be instructive to investigate whether focusing on an increase in the error rate and not focusing on the ratio of training loss to validation loss, would be instructive for working out the right level of training the model, at the same time as avoiding overfitting.

3.6 Image segmentation

The original goal of this research was to be able to use the classified images to describe the numbers and categories of documents in Persian. I investigated using an image segmentation approach. To classify the categories in the data within the images of letters, many approaches could be taken. I first investigated how to do this with labelled data. I found a paper which shows how this can be done. As mentioned, in section 2.2, in each layer outputs channels are created that are half the height and width that allows for a doubling of the number of channels per layer. However in the case of segmentation it is required to end up with an image as the same size as the original. One approach suggests using a convolutional stride of one half instead of two, known as a transpose or deconvolution.[30][31]



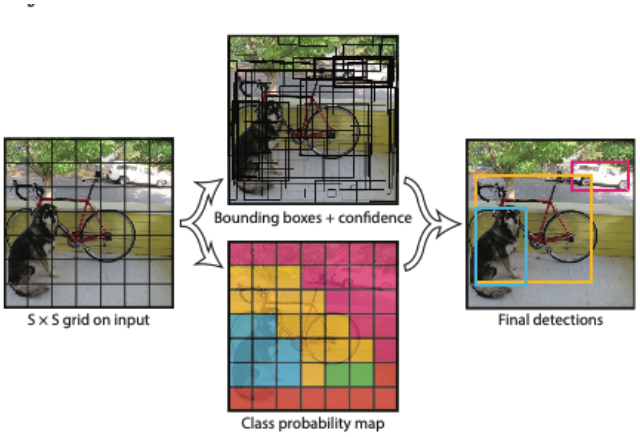
Source: A guide to convolution arithmetic for deep learning [30]

Here there is a 1x1 padding of zeros added to the grid around the outside of the image as well as padding added in between the image pixel. As the 3 x 3 kernel moves around the grid the output will be enlarged from a 3x3 output to a 5 x 5 output. The drawback to this approach is that the white space created doesn't add information, even though it is computationally expensive. Additionally, a 3 x 3 kernel passing over the grid will isolate different areas, some with more coloured pixels than others. This changes the distribution of colour pixels from the original images and therefore the accuracy of information retrieved by the model from each convolutional layer. A better approach could be to increase the size of the images initially by a set scaler value. So if the specified scalar value was sixteen, then a 2x2 image (w,x,y,z), where each pixel was multiplied by sixteen would include, sixteen ws, sixteen xs, sixteen ys, and sixteen zs, which would make the image 8 x 8, ie 16 x 4 = 8 x 8 = 64 pixels. This is called nearest neighbor interpolation.[32] Bilinear interpolation can also be used which takes a weighted average of the pixels

around it to determine the value of the pixel that is being created[33]. However, this approach would need manual labeling for each letter in the letter data set to test the accuracy of the image segmentation.
Two more theoretical approaches were investigated.

3.6.1 Object detection using Bounding boxes.

If the relevant parts of the Letter images could be isolated in bounding boxes, then it might be possible to classify those words, for example the word 'Date' in Persian and by extension the word next to it. The paper 'Scalable Object Detection using Deep Neural Networks' shows how a loss function can be used to match a box to an object within an image and output "a single score for each box, corresponding to its likelihood of containing any object of interest"[34] Since 2014, more advances have been made. A different approach was created using 'regional proposal networks' that, "takes an image as input and outputs a set of rectangular object proposals… (then a) R-CNN detector…uses the proposed regions (to detect objects)."[35]. The recognized YOLO algorithm uses a different approach albeit with some similarities. "(The) network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously."[36]



Source: You Only Look Once: Unified, Real-Time Object Detection [36]

"Each grid cell predicts (the) bounding boxes and confidence scores for those boxes. As can been see in the top image, YOLO predicts multiple bounding boxes per grid cell. At training time (though)… only …one bounding box predictor (is) …responsible for each object. YOLO shares some similarities with R-CNN. Each grid cell proposes potential bounding boxes and scores those boxes using convolutional features. However, YOLO (has) spatial constraints on the grid cell proposals which helps mitigate multiple detections of the same object."[36]

Comparably to YOLO, the SSD: Single Shot MultiBox Detector paper describes a different method to achieve similar performance while also using just one network for all the computation. SSD, "discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales"[37].

Most recently, Focal Loss for Dense Object Detection by Lin et al, strives to combine the world class performance of an R-CNN with the simplicity of SSD and YOLO. It does this by, "(identifying) class imbalance during training as the main obstacle impeding (a) one-stage detector from achieving state-of-

the-art accuracy and propose a new loss function (referred to as 'Focal Loss') that eliminates this barrier." [38]. Lin et al also,"design a simple one-stage object detector called RetinaNet, named for its dense sampling of object locations in an input image." [38]

Further work could be done by using these algorithms, which are implemented in PyTorch[39]. In particular, transfer learning could be used on the Persian data set to train the end layers of the pretrained CNN. By doing this, when the YOLO or other algorithms input the Letter images, bounding boxes could recognise the Persian words such as 'Date' in Persian. This could be the first stage to classify the actual numbers and make sense of them.

A bounding box was created for a Letter image. See code in appendix.

3.6.2 Devise: Deep Visual-Semantic Embedding

An alternative method could involve bringing together the semantics of images, and even words and images. The is uses K-NNs, k nearest neighbours to classify similar looks images and text with each other. The idea then could be to find on the Persian Letters, words that are most similar the trained words such as 'Date' and 'Number' in Persian. See [40]–[42].

4 Results

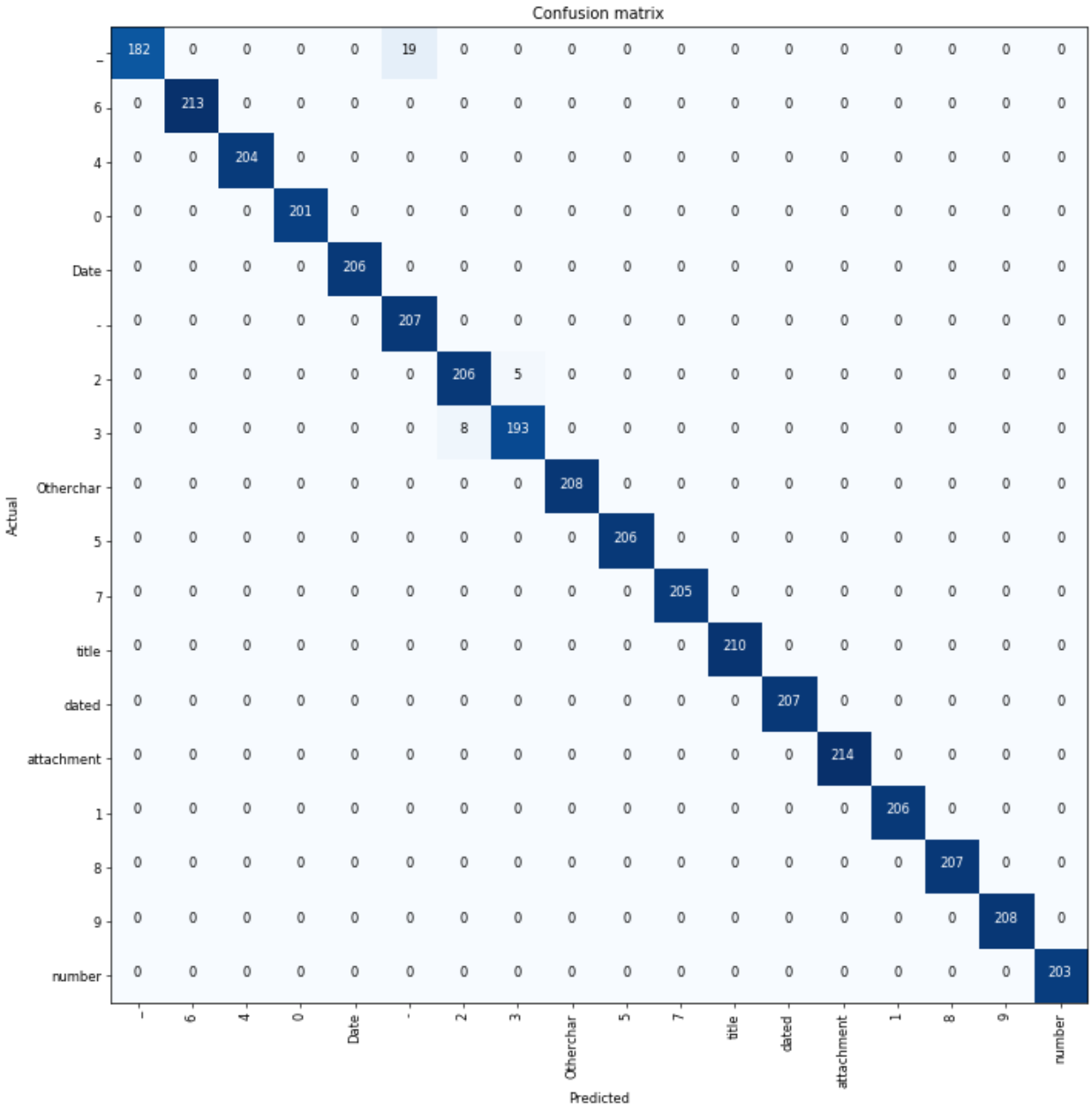
4.1 Persian Image Classification Accuracy

The CNN was trained for four cycles after which the error rate was 0.8 %, which is 99.2% accuracy. Before fine-tuning the model, the model was saved since a set of weights were created through the training that led to this impressive classification accuracy of 99.2%.

The initial results were analysed in more detail. The model was created using a 'learn' object which contains the data and the trained model together, and it is used as a parameter for the Classification Interpretation object. The most common losses were plotted and shown below:

The loss function measures the difference between prediction and ground truth. The higher the confidence in a prediction, that in the end turned out to be false has a higher loss than if the prediction had a lower confidence value attached to it. The figure above shows those images that the algorithm was most confident in its predications but in fact was mistaken. For example, the top left image, was predicted to be a dash but was in fact an underscore, the loss is the third number and was 1.54, and the probability of the predicted class was 0.21; or for example the middle image was predicted to be a three but was in fact a two, in Persian that is, the loss was 1.35 and the probability of the predicted class was 0.26.

A confusion matrix was created. It shows how many times the actual data category was predicted to be that same category, and when an incorrect prediction was made.



It was pretty accurate as is displayed visually. At the same time, it may be hard to get a clear picture of where the predictions were inaccurate. A function called 'most_confused' was used on the interpretation object to display more clearly which categories the model classified inaccurately most often. The result was the following output: [('_', '-', 19), ('3', '2', 8), ('2', '3', 5)]. This output along with the confusion matrix above show that the most confused categories were an underscore with a dash and; the Persian number three with a Persian number two and vice versa. The last number in each block in the array is the number of times each confusion occurred.

Afterwards, I updated the Fast AI code base to the latest version and I ran the same network again. The initial accuracy achieved after 4 epochs was 98% which was lower than before. The categories that were confused were the same as before. However, additional categories were confused as well. [('_', '-', 12),('2',

'3', 11), ('3', '2', 8), ('2', '0', 7), ('6', '2', 5), ('9', '5', 5), ('6', '5', 4), ('dated', 'number', 4), ('-', '_', 3), ('3', '0', 3), ('4', '1', 3), ('5', '6', 3), ('title', 'Date', 3)]. After fine-tuning, the final accuracy rates achieved are the same and are reported below. After investigating further, I found the initial accuracy performance of the updated code to be worse than before the update. I reached out to Jeremy Howard of FastAI to mention this. After investing again even further, I found the accuracy to improve to 99.1%. So, it seems that the performance accuracy is variable by 1-2%.

The fine-tuning in general did improve the accuracy from at its lowest at 98% to 99.7%. Alternatively, the error rate went from 0.025820 to 0.002690 a decrease in the error of 89.5 %. This shows a benefit to fine-tuning by unfreezing the layers after its initial training has been completed where the pre-trained layers are frozen. Further work also could try unfreezing the model completely from the beginning before training.

Also, a model using a Resnet 50 instead of a Resnet 34 was created. I expected the performance to be the even better, as the Resnet 50 has more layers. However, the accuracy was the same as the Resnet 34 after fine-tuning.

5 Discussion

This section will discuss whether the objectives originally set out have been achieved.

Convolutional Neural Networks called Resnets were trained, that achieved 99.7% per cent accuracy on recognising the individual images in the data set. Further work could investigate which types of image data perform better when fine tuning the learning rate. Additional work is needed to be able to translate this learning into recognizing the text that is not is isolated image. It was an objective to be able to Recognise where the data is on the page using deep learning. Although not completed, I investigated possible approaches towards this. Although there are more simple ways using classic image segmentation approaches to segment an image such as SIFT, the objective was to use deep learning to classify where the word data is on the page.

Additional further investigation is needed to see how much the pre-trained weights help for data that is different to the date categories in the pre-trained ImageNet data set. Alternatively one could investigate using a CNN that is pre-trained on a similar data set, for example on the MNIST data set[28], which is an image set of handwritten digits. Instead of using a Resnet, one could try using DenseNets of which there has been research recently showing high levels of accuracy. “The idea of DenseNets is based on the observation that if each layer is directly con- nected to every other layer in a feed-forward fashion then the network will be more accurate and easier to train”[29]. One drawback though is that DenseNets are more memory intensive than Resnets.

In conclusion aspects of the objective of letter and number classification were achieved and a very high level of accuracy was achieved, however, the objective of image segmentation of the Letters was not achieved. As mentioned above, the approach of using bounded boxes object detection and Deep Visual-Semantic Embedding are areas that seem promising to fulfil this objective.

6 Evaluation

This project has been challenging and enjoyable. I have learned a lot. In particular, I gained an in-depth clarity and intuition on the workings of convolutional neural networks and residual neural networks. I also learned how to run a server in the cloud. I also learned about how to use bash and use a server remotely. I learned how to use Python and Pytorch classes and methods and to code them. I learned about the Fast AI library, the various functions and data types, which has given me confidence to be able to utilize deep learning going forward. I learned about how to use jupyter notebooks and the shortcuts that can be useful.

I learned about time management and how it is important especially when undertaking large projects. This is what I would do differently on my next project. I would be disciplined to know exactly what I'm working in each timed period of work. This is because deep learning is very vast in terms of the literature as well as the approaches that can be taken. I found that I was fascinated by all the new advances in deep learning which took up a lot of time learning about them. Instead I should have broken the work up into small pieces and worked on each aspect at a time.

References

[1] I. Kissos and N. Dershowitz, “OCR Error Correction Using Character Correction and Feature-Based Word Classification,” in *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*, Santorini, Greece, 2016, pp. 198–203.

[2] “The Persian Language.” .

[3] C. Sonia Yousfi, “Embedded Arabic text detection and recognition in videos,” PhD Thesis, INSA de Lyon, 2016.

[4] “PyTorch.” [Online]. Available: https://www.pytorch.org. [Accessed: 10-Jan-2019].

[5] “fast.ai,” *GitHub*. [Online]. Available: https://github.com/fastai. [Accessed: 10-Jan-2019].

[6] “An Even Easier Introduction to CUDA,” *NVIDIA Developer Blog*, 25-Jan-2017. [Online]. Available: https://devblogs.nvidia.com/even-easier-introduction-cuda/. [Accessed: 10-Jan-2019].

[7] M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” in *Computer Vision – ECCV 2014*, vol. 8689, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 818–833.

[8] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *ArXiv151203385 Cs*, Dec. 2015.

[9] “CS231n Convolutional Neural Networks for Visual Recognition.” [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Accessed: 04-Jan-2019].

[10] D. Han, Q. Liu, and W. Fan, “A new image classification method using CNN transfer learning and web data augmentation,” *Expert Syst. Appl.*, vol. 95, pp. 43–56, Apr. 2018.

[11] “Stanford DAWN Deep Learning Benchmark (DAWNBench) ·.” [Online]. Available: https://dawn.cs.stanford.edu/benchmark/. [Accessed: 07-Jan-2019].

[12] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, “Visualizing the Loss Landscape of Neural Nets,” *ArXiv171209913 Cs Stat*, Dec. 2017.

[13] “AdaptiveAvgPool2d.” [Online]. Available: https://pytorch.org/docs/stable/nn.html#torch.nn.AdaptiveAvgPool2d. [Accessed: 03-Jan-2019].

[14] “Training a Classifier — PyTorch Tutorials 1.0.0.dev20181228 documentation.” [Online]. Available: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html. [Accessed: 03-Jan-2019].

[15] “nn package — PyTorch Tutorials 1.0.0.dev20181228 documentation.” [Online]. Available: https://pytorch.org/tutorials/beginner/former_torchies/nn_tutorial.html. [Accessed: 04-Jan-2019].

[16] “callbacks.hooks | fastai.” [Online]. Available: https://docs.fast.ai/callbacks.hooks.html#hook_output. [Accessed: 04-Jan-2019].

[17] “vision.data | fastai.” [Online]. Available: https://docs.fast.ai/vision.data.html#ImageDataBunch. [Accessed: 04-Jan-2019].

[18] 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.

[19] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?,” p. 9.

[20] “Memorizing is not learning! — 6 tricks to prevent overfitting in machine learning.” [Online]. Available: https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42. [Accessed: 06-Jan-2019].
[21] L. N. Smith, “A disciplined approach to neural network hyper-parameters: Part 1 -- learning rate, batch size, momentum, and weight decay,” *ArXiv180309820 Cs Stat*, Mar. 2018.
[22] J. Howard and S. Ruder, “Universal Language Model Fine-tuning for Text Classification,” *ArXiv180106146 Cs Stat*, Jan. 2018.
[23] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima,” *ArXiv160904836 Cs Math*, Sep. 2016.
[24] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio, “Sharp Minima Can Generalize For Deep Nets,” *ArXiv170304933 Cs*, Mar. 2017.
[25] E. Hoffer, I. Hubara, and D. Soudry, “Train longer, generalize better: closing the generalization gap in large batch training of neural networks,” *ArXiv170508741 Cs Stat*, May 2017.
[26] L. N. Smith and N. Topin, “Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates,” *ArXiv170807120 Cs Stat*, Aug. 2017.
[27] “basic_train | fastai.” [Online]. Available: https://docs.fast.ai/basic_train.html#Learner.lr_range. [Accessed: 08-Jan-2019].
[28] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010.
[29] S. Jégou, M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio, “The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation,” *ArXiv161109326 Cs*, Nov. 2016.
[30] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” *ArXiv160307285 Cs Stat*, Mar. 2016.
[31] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” *ArXiv150504597 Cs*, May 2015.
[32] R. Olivier and C. Hanqiang, “Nearest Neighbor Value Interpolation,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 3, no. 4, 2012.
[33] P. Getreuer, “Linear Methods for Image Interpolation,” *IPOL J.*, vol. 1, 2011.
[34] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, “Scalable Object Detection Using Deep Neural Networks,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA, 2014, pp. 2155–2162.
[35] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.
[36] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 779–788.
[37] W. Liu *et al.*, “SSD: Single Shot MultiBox Detector,” in *Computer Vision – ECCV 2016*, vol. 9905, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 21–37.
[38] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, “Focal Loss for Dense Object Detection,” p. 10.

[39] “How to implement a YOLO (v3) object detector from scratch in PyTorch: Part 2.” [Online]. Available: https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-2/. [Accessed: 10-Jan-2019].
[40] D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” *ArXiv14090473 Cs Stat*, Sep. 2014.
[41] A. Frome *et al.*, “DeViSE: A Deep Visual-Semantic Embedding Model,” p. 9.
[42] Y. Wu *et al.*, “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation,” *ArXiv160908144 Cs*, Sep. 2016.

Appendix

Renaming file names

Renaming.py

```
# os is a library that gives us the ability to make OS changes
import os


def name_file_numbers(directory, string, integer):
    # iterate over every file name in the directory
    print (directory)
    file_number = startcount
    for file_name in os.listdir(directory):
        file_number = file_number + 1
        extension = ".jpg"
        file_name2 = objectname+"."
        new_file_name = '%s/%s%02d%s' % (directory, file_name2, file_number, extension)
        old_file_name = '%s/%s' % (directory, file_name)
        print (old_file_name)
        print (new_file_name)
        # rename the file
        os.rename(old_file_name, new_file_name)

path = Path("/Users/marckendal/Dropbox/Dissertation New/Data/ocrdata copy/")

files = os.listdir(path)
for name in files:
    print(name)


foldernames = []
def listdir_nohidden(path):
    for f in os.listdir(path):
        if not f.startswith('.'):
            foldernames.append(f)


for foldername in foldernames :
      PATHfolder = os.path.abspath('/Users/marckendal/Dropbox/Dissertation New/Data/ocrdata copy/'+
foldername)
    objectname = foldername
    startcount = 0
    name_file_numbers(PATHfolder, objectname, startcount)
```

Tar Decompression

```
import tarfile
tar = tarfile.open("data/SplitData7Tar.tgz")
```

```
tar.extractall()
tar.close()
```

Bounding Box Creation

```
%reload_ext autoreload
%autoreload 2
%matplotlib inline


from fastai import *
from fastai.vision import *


import zipfile
with zipfile.ZipFile("data/LETTER.zip","r") as zip_ref:
    zip_ref.extractall("data")

import pathlib
path = Path("LETTER/letter (1).jpg")

img = open_image(path)
bbox = ImageBBox.create(500,500, [[0, 70, 70, 200]], labels=[0], classes=['Area of Interest'])
img.show(y=bbox)
```

Move Data to Remote Server

```
scp -rp fastai/course-v3-master paperspace@184.105.157.160:/home/paperspace/fastai/
```

Persian Convolutional Kernels & Heatmap

In [1]:
```
%reload_ext autoreload
%autoreload 2
%matplotlib inline

from fastai.vision import *
```

In [2]:
```
bs = 64
```

In [3]:
```
import tarfile
tar = tarfile.open("data/SplitData7Tar.tgz")
tar.extractall()
tar.close()

import pathlib
path = Path("SplitData7")
```

Data augmentation

In [4]:
```
tfms = get_transforms(do_flip=False)
```

In [5]:
```
doc(get_transforms)
```

In [6]:
```
src = ImageItemList.from_folder(path).random_split_by_pct(0.2, seed=2)
```

In [7]:
```
def get_data(size, bs, padding_mode='reflection'):
    return (src.label_from_folder()
           .transform(tfms, size=size, padding_mode=padding_mode)
           .databunch(bs=bs).normalize(imagenet_stats))
```

In [8]:
```
data = get_data(224, bs, 'zeros')
```

In [99]:
```
def _plot(i,j,ax):
    x,y = data.train_ds[8888]
    x.show(ax, y=y)

plot_multi(_plot, 3, 3, figsize=(8,8))
```

In [59]:
```
data
```

Out[59]:
```
ImageDataBunch;

Train: LabelList
y: CategoryList (16632 items)
[Category test, Category test, Category test, Category test, Categor
y test]...
Path: SplitData7
x: ImageItemList (16632 items)
[Image (3, 50, 90), Image (3, 50, 90), Image (3, 50, 90), Image (3,
50, 90), Image (3, 50, 90)]...
Path: SplitData7;

Valid: LabelList
y: CategoryList (4158 items)
[Category 6, Category 6, Category 6, Category Otherchar, Category 0]
...
Path: SplitData7
x: ImageItemList (4158 items)
[Image (3, 50, 90), Image (3, 50, 90), Image (3, 50, 90), Image (3,
50, 90), Image (3, 50, 90)]...
Path: SplitData7;

Test: None
```

In [60]:
```
data = get_data(224,bs)
```

In [61]:
```
plot_multi(_plot, 3, 3, figsize=(8,8))
```

Train a model

In [62]:
```
gc.collect()
learn = create_cnn(data, models.resnet34, metrics=error_rate, bn_final
=True)
```

In [63]:
```
learn.fit_one_cycle(3, slice(1e-2), pct_start=0.8)
```

Total time: 03:08

epoch	train_loss	valid_loss	error_rate
1	1.974069	1.383022	0.325397
2	1.105415	0.861188	0.214286
3	0.844862	0.678317	0.194324

In [64]:
```
learn.unfreeze()
learn.fit_one_cycle(2, max_lr=slice(1e-6,1e-3), pct_start=0.8)
```

Total time: 02:11

epoch	train_loss	valid_loss	error_rate
1	0.790859	0.644937	0.193362
2	0.755142	0.605667	0.189274

In [65]:
```
data = get_data(352,bs)
learn.data = data
```

In [66]:
```
learn.fit_one_cycle(2, max_lr=slice(1e-6,1e-4))
```

Total time: 04:55

epoch	train_loss	valid_loss	error_rate
1	0.754411	0.605008	0.188071
2	0.713981	0.602794	0.188552

In [67]:
```
learn.save('352')
```

Convolution kernel

In [68]:
```
data = get_data(352,16)
```

In [69]:
```
data
```

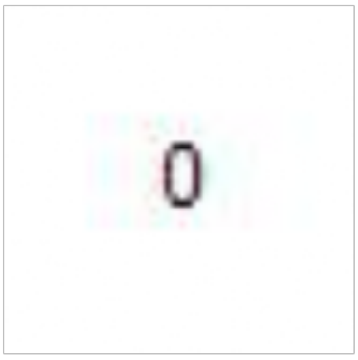
Out[69]:
```
ImageDataBunch;

Train: LabelList
y: CategoryList (16632 items)
[Category test, Category test, Category test, Category test, Categor
y test]...
Path: SplitData7
x: ImageItemList (16632 items)
[Image (3, 50, 90), Image (3, 50, 90), Image (3, 50, 90), Image (3,
50, 90), Image (3, 50, 90)]...
Path: SplitData7;

Valid: LabelList
y: CategoryList (4158 items)
[Category 6, Category 6, Category 6, Category Otherchar, Category 0]
...
Path: SplitData7
x: ImageItemList (4158 items)
[Image (3, 50, 90), Image (3, 50, 90), Image (3, 50, 90), Image (3,
50, 90), Image (3, 50, 90)]...
Path: SplitData7;

Test: None
```

In [70]:
```
learn = create_cnn(data, models.resnet34, metrics=error_rate)
```

In [101]:
```
x,y = data.valid_ds[4000]
x.show()
data.valid_ds.y[100]
```

```
Clipping input data to the valid range for imshow with RGB data ([0.
.1] for floats or [0..255] for integers).
```

Out[101]:
```
Category test
```



In [102]:
```
idx=14
x,y = data.valid_ds[400]
x.show()
data.valid_ds.y[idx]
```

```
Clipping input data to the valid range for imshow with RGB data ([0.
.1] for floats or [0..255] for integers).
```

Out[102]: Category Date

In [103]:
```
k = tensor([
    [0.  ,-5/3,1],
    [-5/3,-5/3,1],
    [1.  ,1   ,1],
]).expand(1,3,3,3)/6
```

In [104]:
```
from fastai.callbacks.hooks import *
```

In [105]:
```
k
```

Out[105]:
```
tensor([[[[ 0.0000, -0.2778,  0.1667],
          [-0.2778, -0.2778,  0.1667],
          [ 0.1667,  0.1667,  0.1667]],

         [[ 0.0000, -0.2778,  0.1667],
          [-0.2778, -0.2778,  0.1667],
          [ 0.1667,  0.1667,  0.1667]],

         [[ 0.0000, -0.2778,  0.1667],
          [-0.2778, -0.2778,  0.1667],
          [ 0.1667,  0.1667,  0.1667]]]])
```

In [106]:
```
k.shape
```

Out[106]:
```
torch.Size([1, 3, 3, 3])
```

In [112]:
```
t = data.valid_ds[400][0].data; t.shape
```

Out[112]:
```
torch.Size([3, 352, 352])
```

In [113]:
```
t[None].shape
```

Out[113]:
```
torch.Size([1, 3, 352, 352])
```

In [114]:
```
edge = F.conv2d(t[None], k)
```

In [115]:
```
show_image(edge[0], figsize=(5,5));
```

In [116]:
```
data.c
```

Out[116]:
```
19
```

In [117]:
```
learn.model
```

Out[117]:
```
Sequential(
  (0): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3
, 3), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
    (2): ReLU(inplace)
    (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, c
eil_mode=False)
    (4): Sequential(
      (0): BasicBlock(
```

```
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (5): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bi
as=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        )
```

```
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
      )
      (3): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
      )
    )
    (6): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), b
ias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True
```

```
, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
      )
      (3): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
      )
      (4): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
      )
      (5): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
```

```
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
      )
    )
    (7): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), b
ias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True
, track_running_stats=True)
      )
    )
```

```
  )
  (1): Sequential(
    (0): AdaptiveConcatPool2d(
      (ap): AdaptiveAvgPool2d(output_size=1)
      (mp): AdaptiveMaxPool2d(output_size=1)
    )
    (1): Lambda()
    (2): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
    (3): Dropout(p=0.25)
    (4): Linear(in_features=1024, out_features=512, bias=True)
    (5): ReLU(inplace)
    (6): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
    (7): Dropout(p=0.5)
    (8): Linear(in_features=512, out_features=19, bias=True)
  )
)
```

In [118]:
```
learn.summary()
```

```
==========================================================================
==
Layer (type)         Output Shape         Param #    Trainable
==========================================================================
==
Conv2d               [16, 64, 176, 176]   9408       False
__________________________________________________________________________
__
BatchNorm2d          [16, 64, 176, 176]   128        True
__________________________________________________________________________
__
ReLU                 [16, 64, 176, 176]   0          False
__________________________________________________________________________
__
MaxPool2d            [16, 64, 88, 88]     0          False
__________________________________________________________________________
__
Conv2d               [16, 64, 88, 88]     36864      False
__________________________________________________________________________
__
BatchNorm2d          [16, 64, 88, 88]     128        True
__________________________________________________________________________
__
ReLU                 [16, 64, 88, 88]     0          False
__________________________________________________________________________
__
Conv2d               [16, 64, 88, 88]     36864      False
__________________________________________________________________________
__
```

BatchNorm2d	[16, 64, 88, 88]	128	True
__			
Conv2d	[16, 64, 88, 88]	36864	False
__			
BatchNorm2d	[16, 64, 88, 88]	128	True
__			
ReLU	[16, 64, 88, 88]	0	False
__			
Conv2d	[16, 64, 88, 88]	36864	False
__			
BatchNorm2d	[16, 64, 88, 88]	128	True
__			
Conv2d	[16, 64, 88, 88]	36864	False
__			
BatchNorm2d	[16, 64, 88, 88]	128	True
__			
ReLU	[16, 64, 88, 88]	0	False
__			
Conv2d	[16, 64, 88, 88]	36864	False
__			
BatchNorm2d	[16, 64, 88, 88]	128	True
__			
Conv2d	[16, 128, 44, 44]	73728	False
__			
BatchNorm2d	[16, 128, 44, 44]	256	True
__			
ReLU	[16, 128, 44, 44]	0	False
__			
Conv2d	[16, 128, 44, 44]	147456	False
__			
BatchNorm2d	[16, 128, 44, 44]	256	True
__			
Conv2d	[16, 128, 44, 44]	8192	False

BatchNorm2d	[16, 128, 44, 44]	256	True
Conv2d	[16, 128, 44, 44]	147456	False
BatchNorm2d	[16, 128, 44, 44]	256	True
ReLU	[16, 128, 44, 44]	0	False
Conv2d	[16, 128, 44, 44]	147456	False
BatchNorm2d	[16, 128, 44, 44]	256	True
Conv2d	[16, 128, 44, 44]	147456	False
BatchNorm2d	[16, 128, 44, 44]	256	True
ReLU	[16, 128, 44, 44]	0	False
Conv2d	[16, 128, 44, 44]	147456	False
BatchNorm2d	[16, 128, 44, 44]	256	True
Conv2d	[16, 128, 44, 44]	147456	False
BatchNorm2d	[16, 128, 44, 44]	256	True
ReLU	[16, 128, 44, 44]	0	False
Conv2d	[16, 128, 44, 44]	147456	False
BatchNorm2d	[16, 128, 44, 44]	256	True
Conv2d	[16, 256, 22, 22]	294912	False

BatchNorm2d	[16, 256, 22, 22]	512	True
ReLU	[16, 256, 22, 22]	0	False
Conv2d	[16, 256, 22, 22]	589824	False
BatchNorm2d	[16, 256, 22, 22]	512	True
Conv2d	[16, 256, 22, 22]	32768	False
BatchNorm2d	[16, 256, 22, 22]	512	True
Conv2d	[16, 256, 22, 22]	589824	False
BatchNorm2d	[16, 256, 22, 22]	512	True
ReLU	[16, 256, 22, 22]	0	False
Conv2d	[16, 256, 22, 22]	589824	False
BatchNorm2d	[16, 256, 22, 22]	512	True
Conv2d	[16, 256, 22, 22]	589824	False
BatchNorm2d	[16, 256, 22, 22]	512	True
ReLU	[16, 256, 22, 22]	0	False
Conv2d	[16, 256, 22, 22]	589824	False
BatchNorm2d	[16, 256, 22, 22]	512	True

Conv2d	[16, 256, 22, 22]	589824	False
BatchNorm2d	[16, 256, 22, 22]	512	True
ReLU	[16, 256, 22, 22]	0	False
Conv2d	[16, 256, 22, 22]	589824	False
BatchNorm2d	[16, 256, 22, 22]	512	True
Conv2d	[16, 256, 22, 22]	589824	False
BatchNorm2d	[16, 256, 22, 22]	512	True
ReLU	[16, 256, 22, 22]	0	False
Conv2d	[16, 256, 22, 22]	589824	False
BatchNorm2d	[16, 256, 22, 22]	512	True
Conv2d	[16, 256, 22, 22]	589824	False
BatchNorm2d	[16, 256, 22, 22]	512	True
ReLU	[16, 256, 22, 22]	0	False
Conv2d	[16, 256, 22, 22]	589824	False
BatchNorm2d	[16, 256, 22, 22]	512	True
Conv2d	[16, 512, 11, 11]	1179648	False
BatchNorm2d	[16, 512, 11, 11]	1024	True

ReLU	[16, 512, 11, 11]	0	False
Conv2d	[16, 512, 11, 11]	2359296	False
BatchNorm2d	[16, 512, 11, 11]	1024	True
Conv2d	[16, 512, 11, 11]	131072	False
BatchNorm2d	[16, 512, 11, 11]	1024	True
Conv2d	[16, 512, 11, 11]	2359296	False
BatchNorm2d	[16, 512, 11, 11]	1024	True
ReLU	[16, 512, 11, 11]	0	False
Conv2d	[16, 512, 11, 11]	2359296	False
BatchNorm2d	[16, 512, 11, 11]	1024	True
Conv2d	[16, 512, 11, 11]	2359296	False
BatchNorm2d	[16, 512, 11, 11]	1024	True
ReLU	[16, 512, 11, 11]	0	False
Conv2d	[16, 512, 11, 11]	2359296	False
BatchNorm2d	[16, 512, 11, 11]	1024	True
AdaptiveAvgPool2d	[16, 512, 1, 1]	0	False
AdaptiveMaxPool2d	[16, 512, 1, 1]	0	False

```
__
Lambda                   [16, 1024]            0          False
__
BatchNorm1d              [16, 1024]            2048       True
__
Dropout                  [16, 1024]            0          False
__
Linear                   [16, 512]             524800     True
__
ReLU                     [16, 512]             0          False
__
BatchNorm1d              [16, 512]             1024       True
__
Dropout                  [16, 512]             0          False
__
Linear                   [16, 19]              9747       True
__

Total params:  21822291
Total trainable params:  554643
Total non-trainable params:  21267648
```

Heatmap

In [119]:
```
m = learn.model.eval();
```

In [120]:
```
xb,_ = data.one_item(x)
xb_im = Image(data.denorm(xb)[0])
xb = xb.cuda()
```

In [121]:
```
from fastai.callbacks.hooks import *
```

In [122]:
```
def hooked_backward(cat=y):
    with hook_output(m[0]) as hook_a:
        with hook_output(m[0], grad=True) as hook_g:
            preds = m(xb)
            preds[0,int(cat)].backward()
    return hook_a,hook_g
```

In [123]:
```
hook_a,hook_g = hooked_backward()
```

In [124]:
```
acts  = hook_a.stored[0].cpu()
acts.shape
```

Out[124]:
```
torch.Size([512, 11, 11])
```

In [125]:
```
avg_acts = acts.mean(0)
avg_acts.shape
```

Out[125]:
```
torch.Size([11, 11])
```

In [126]:
```
def show_heatmap(hm):
    _,ax = plt.subplots()
    xb_im.show(ax)
    ax.imshow(hm, alpha=0.6, extent=(0,352,352,0),
              interpolation='bilinear', cmap='magma');
```

In [127]:
```
show_heatmap(avg_acts)
```

```
Clipping input data to the valid range for imshow with RGB data ([0.
.1] for floats or [0..255] for integers).
```

Persian Classification

In [1]:
```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

In [2]:
```
from fastai import *
from fastai.vision import *
```

In [3]:
```
bs = 64
```

Data exploration

In [4]:
```
import pathlib
path = Path("SplitData7")
```

In [5]:
```
import tarfile
tar = tarfile.open("data/SplitData7Tar.tgz")
tar.extractall()
tar.close()
```

In [6]:
```
path.ls()
```

Out[6]:
```
[PosixPath('SplitData7/.DS_Store'),
 PosixPath('SplitData7/test'),
 PosixPath('SplitData7/._test'),
 PosixPath('SplitData7/._.DS_Store'),
 PosixPath('SplitData7/._train'),
 PosixPath('SplitData7/valid'),
 PosixPath('SplitData7/models'),
 PosixPath('SplitData7/train'),
 PosixPath('SplitData7/._valid')]
```

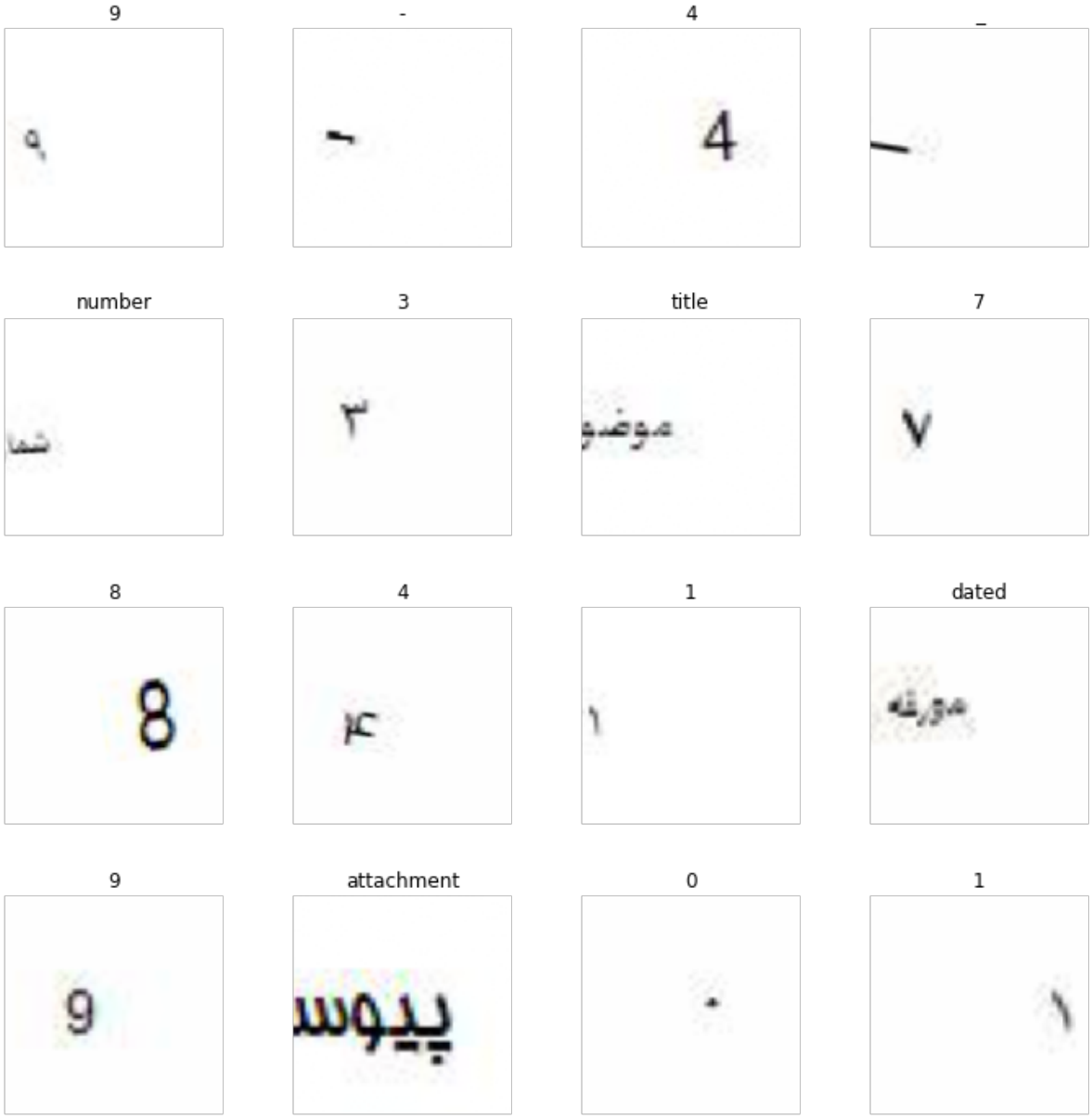
In [7]:
```
tfms = get_transforms(do_flip=False)
```

In [8]:
```
data = ImageDataBunch.from_folder(path, ds_tfms=tfms, size=224,).norma
lize()
```

In [9]:
```
data.batch_stats()
```

Out[9]:
```
[tensor([-0.0002, -0.0002, -0.0002]), tensor([1., 1., 1.])]
```

In [10]:
```
data.show_batch(rows=4, figsize=(10,10))
```



In [11]:
```
print(data.classes)
len(data.classes),data.c
```

```
['-', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'Date', 'Oth
erchar', '_', 'attachment', 'dated', 'number', 'title']
```

Out[11]:
```
(18, 18)
```

Resnet 34

We will train for 4 epochs ie 4 cycles through all our data.

In [12]:
```
learn = create_cnn(data, models.resnet34, metrics=error_rate)
```

In [13]:
```
learn.fit_one_cycle(4)
```

Total time: 04:28

epoch	train_loss	valid_loss	error_rate
1	1.354882	0.619360	0.200914
2	0.655668	0.244712	0.076116
3	0.383952	0.105953	0.032813
4	0.297078	0.090897	0.025282

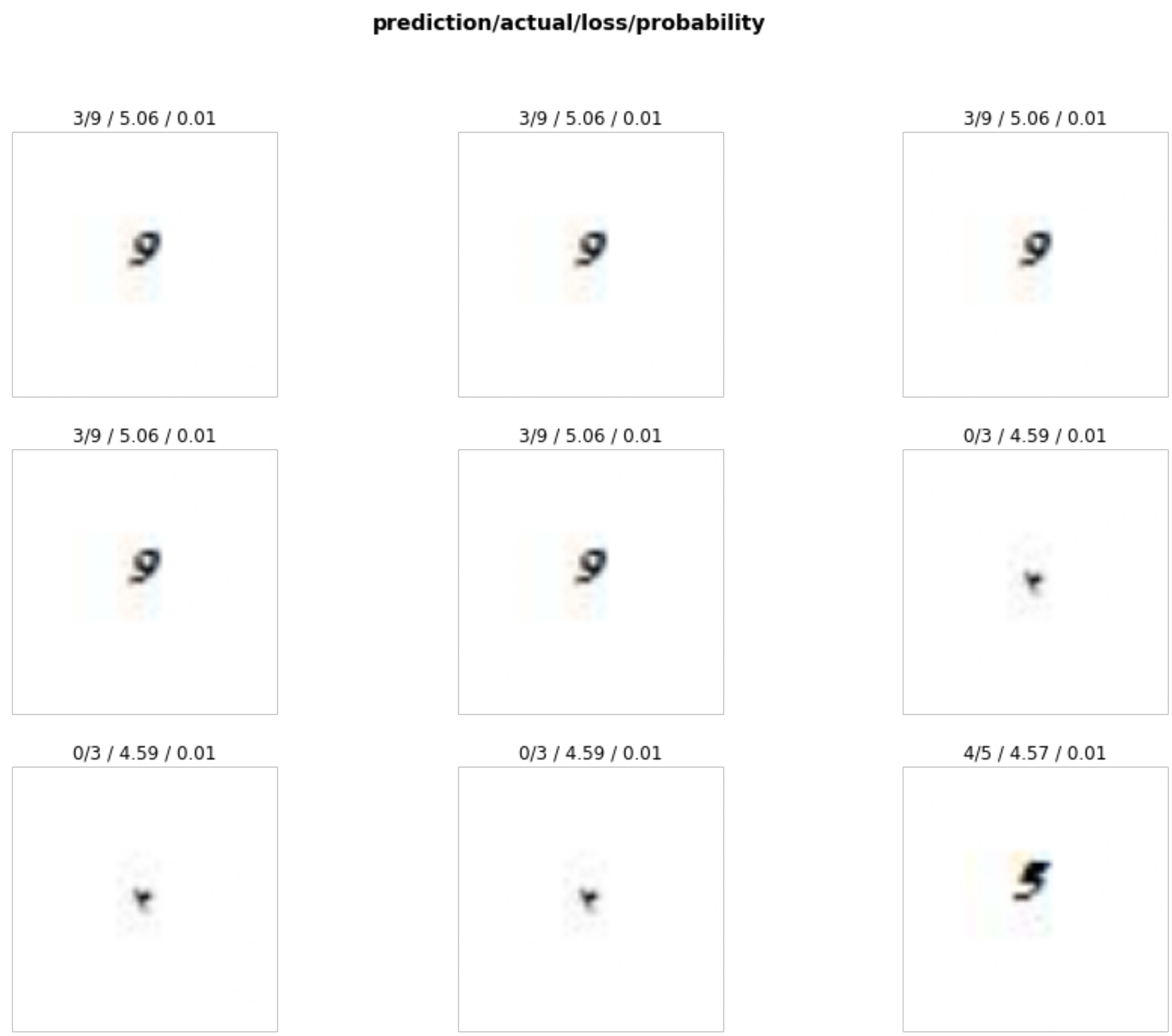
In [14]:
```
learn.save('stage-1')
```

Results

In [15]:
```
interp = ClassificationInterpretation.from_learner(learn)
```

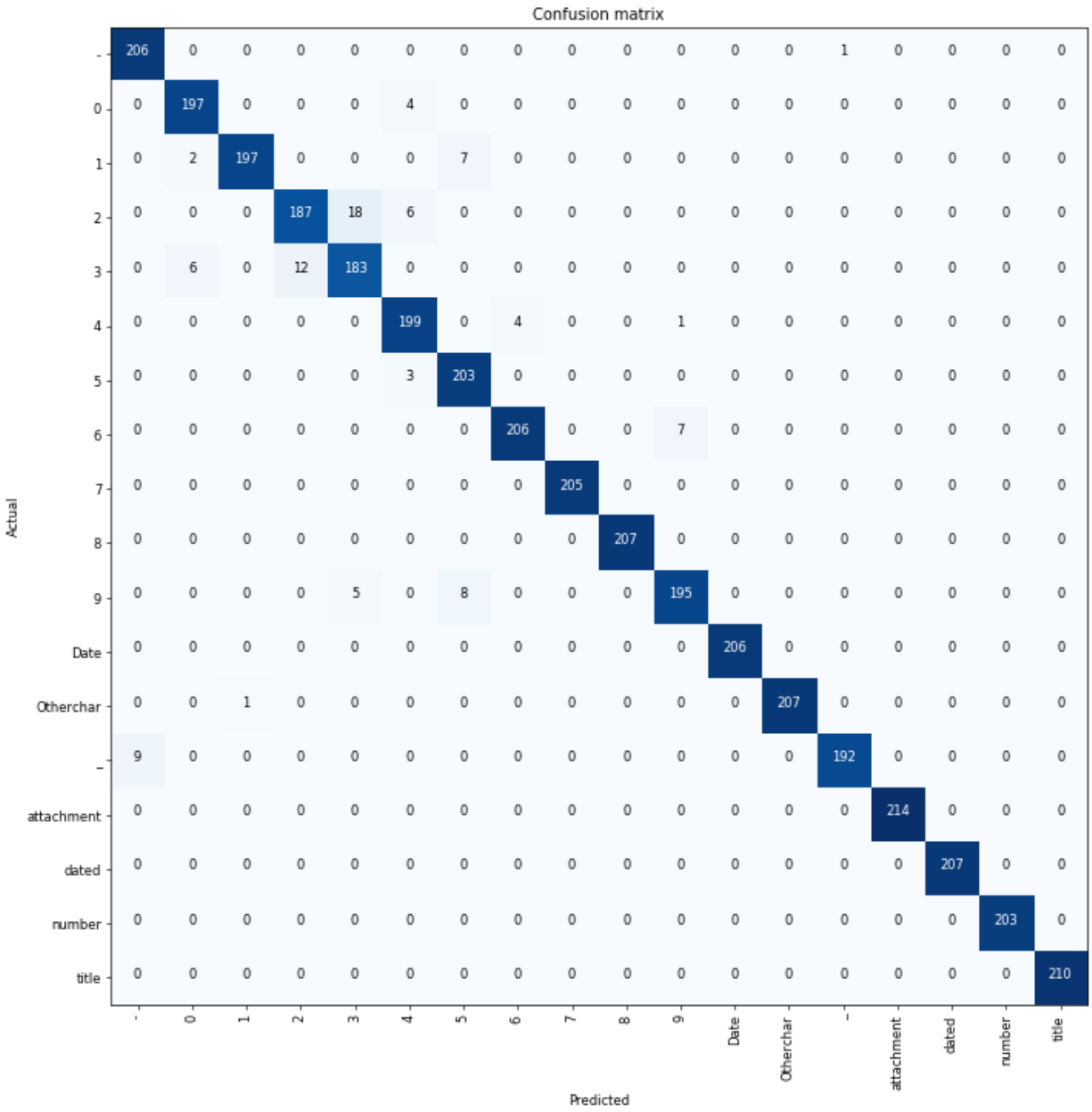
In [16]:
```
interp.plot_top_losses(9, figsize=(15,11))
```

```
Clipping input data to the valid range for imshow with RGB data ([0.
.1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0.
.1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0.
.1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0.
.1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0.
.1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0.
.1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0.
.1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0.
.1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0.
.1] for floats or [0..255] for integers).
```



In [17]:
```
doc(interp.plot_top_losses)
```

In [18]:
```
interp.plot_confusion_matrix(figsize=(12,12), dpi=60)
```



In [19]:
```
interp.most_confused(min_val=2)
```

Out[19]:
```
[('2', '3', 18),
 ('3', '2', 12),
 ('_', '-', 9),
 ('9', '5', 8),
 ('1', '5', 7),
 ('6', '9', 7),
 ('2', '4', 6),
 ('3', '0', 6),
 ('9', '3', 5),
 ('0', '4', 4),
 ('4', '6', 4),
 ('5', '4', 3)]
```

Finetuning and Unfreezing

In [20]:
```
learn.unfreeze()
```

In [21]:
```
learn.fit_one_cycle(5)
```

Total time: 06:18

epoch	train_loss	valid_loss	error_rate
1	0.199353	0.090402	0.034965
2	0.135889	0.172604	0.047606
3	0.080019	0.158880	0.038462
4	0.047095	0.009239	0.004572
5	0.032292	0.006659	0.003228

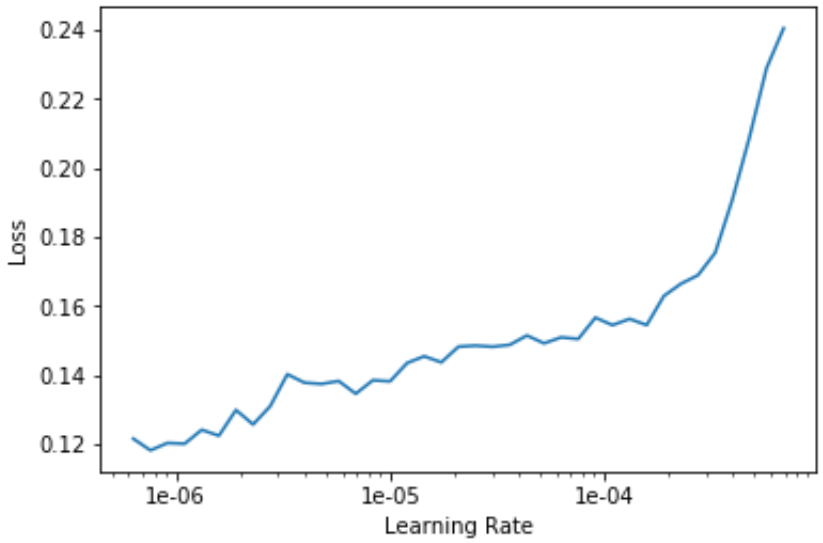
In [22]:
```
learn.save('stage-34-2')
```

In [ ]:
```
learn.load('stage-1');
```

In [26]:
```
learn.lr_find()
```

```
LR Finder is complete, type {learner_name}.recorder.plot() to see th
e graph.
```

In [27]:
```
learn.recorder.plot()
```



In [28]:
```
learn.unfreeze()
learn.fit_one_cycle(5, max_lr=slice(1e-6,1e-4))
```

Total time: 04:52

epoch	train_loss	valid_loss	error_rate
1	0.147511	0.023580	0.008876
2	0.109596	0.013737	0.005379
3	0.082692	0.014603	0.007531
4	0.065604	0.010105	0.003228
5	0.067824	0.010046	0.003228

In [29]:
```
learn.load('stage-1');
```

In [30]:
```
learn.unfreeze()
learn.fit_one_cycle(5, max_lr=slice(1e-6,1e-3))
```

Total time: 04:52

epoch	train_loss	valid_loss	error_rate
1	0.143642	0.036821	0.014793
2	0.105953	0.030599	0.010221
3	0.071585	0.013625	0.005917
4	0.052615	0.011144	0.006455
5	0.044857	0.008341	0.003228

In [31]:
```
learn.load('stage-1');
```

In [32]:
```
learn.unfreeze()
learn.fit_one_cycle(5, max_lr=slice(1e-5,1e-3))
```

Total time: 04:56

epoch	train_loss	valid_loss	error_rate
1	0.157370	0.055645	0.020979
2	0.112620	0.016552	0.006186
3	0.071357	0.013187	0.005648
4	0.042194	0.006722	0.004841
5	0.028892	0.005923	0.003228

In [33]:
```
learn.load('stage-1');
```

In [34]:
```
learn.unfreeze()
learn.fit_one_cycle(5, max_lr=slice(1e-5,1e-4))
```

Total time: 04:51

epoch	train_loss	valid_loss	error_rate
1	0.128170	0.032906	0.012372
2	0.090137	0.012434	0.003765
3	0.065663	0.008648	0.003228
4	0.047594	0.006792	0.003228
5	0.041172	0.005955	0.003228

Resnet 50

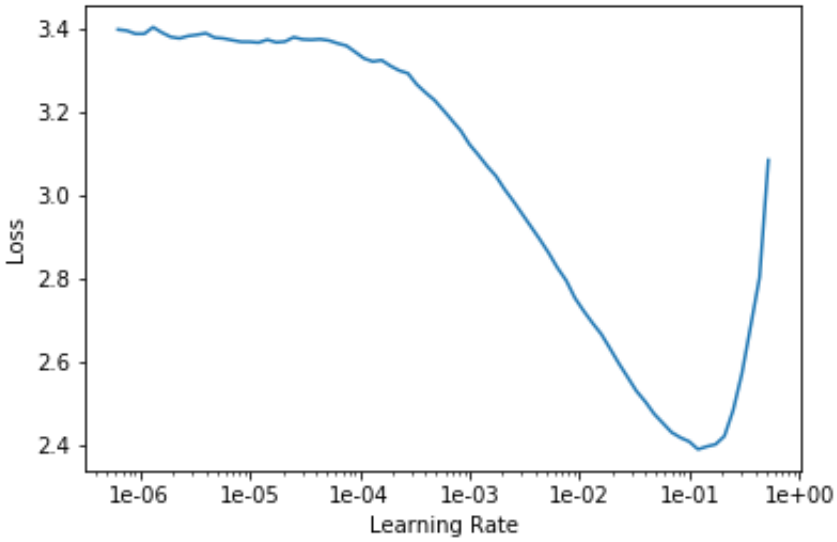
It might be needed to restart the kernel if the notebook is out of memory. If so run until the end of the data exploration from the beginning of the notebook before running this code. Alternatively, just replace the First line of code inorder to specify a lower batch number in this case bs=32 : data = ImageDataBunch.from_folder(path,ds_tfms=tfms, size=299, bs=32).normalize()

In [13]:
```
data = ImageDataBunch.from_folder(path,ds_tfms=tfms,
                                  size=299).normalize()
```

In [14]:
```
learn = create_cnn(data, models.resnet50, metrics=error_rate)
```

In [15]:
```
learn.lr_find()
learn.recorder.plot()
```

```
LR Finder is complete, type {learner_name}.recorder.plot() to see th
e graph.
```



In [16]:
```
learn.fit_one_cycle(8)
```

Total time: 24:45

epoch	train_loss	valid_loss	error_rate
1	1.396531	0.821648	0.290748
2	0.681264	0.362030	0.121302
3	0.321679	0.134769	0.047068
4	0.185383	0.045111	0.018020
5	0.117454	0.020607	0.005110
6	0.091293	0.019298	0.010221
7	0.072667	0.016700	0.006186
8	0.060499	0.011509	0.005110

In [17]:
```
learn.save('stage-1-50')
```

In [17]:
```
learn.load('stage-1-50');
```

In [18]:
```
learn.unfreeze()
learn.fit_one_cycle(3, max_lr=slice(1e-6,1e-4))
```

Total time: 11:15

epoch	train_loss	valid_loss	error_rate
1	0.054200	0.013474	0.004841
2	0.046045	0.010398	0.004572
3	0.038174	0.007728	0.004572

In [19]:
```
learn.load('stage-1-50');
```

In [20]:
```
learn.unfreeze()
learn.fit_one_cycle(3, max_lr=slice(1e-5,1e-2))
```

Total time: 11:10

epoch	train_loss	valid_loss	error_rate
1	0.284041	1.144491	0.291555
2	0.077128	0.017355	0.008069
3	0.031069	0.009920	0.003228

In [21]:
```
learn.load('stage-1-50');
```

In [22]:
```
learn.unfreeze()
learn.fit_one_cycle(3, max_lr=slice(1e-5,1e-1))
```

Total time: 11:11

epoch	train_loss	valid_loss	error_rate
1	1.891478	260.065094	0.480635
2	0.663226	0.179438	0.044110
3	0.114611	0.026968	0.009952

In [23]:
```
learn.load('stage-1-50');
```

In [24]:
```
learn.unfreeze()
learn.fit_one_cycle(3, max_lr=slice(1e-4,1e-2))
```

Total time: 11:12

epoch	train_loss	valid_loss	error_rate
1	0.362691	0.542728	0.144432
2	0.102549	0.020921	0.006724
3	0.030937	0.010374	0.004303

In [25]:
```
learn.load('stage-1-50');
```

In [26]:
```
learn.unfreeze()
learn.fit_one_cycle(3, max_lr=slice(1e-4,1e-1))
```

Total time: 11:11

epoch	train_loss	valid_loss	error_rate
1	1.945657	219.030380	0.827864
2	0.839811	1.437172	0.089833
3	0.111149	0.020919	0.008069

In [27]:
```
learn.load('stage-1-50');
```

In [18]:
```
learn.unfreeze()
learn.fit_one_cycle(3)
```

Total time: 12:09

epoch	train_loss	valid_loss	error_rate
1	0.210617	0.314152	0.104895
2	0.060381	0.027693	0.010221
3	0.024746	0.007659	0.002690

In [19]:
```
learn.save('stage-2-50')
```

In [30]:
```
learn.load('stage-2-50');
```

In [31]:
```
interp = ClassificationInterpretation.from_learner(learn)
```

In [32]:
```
interp.most_confused(min_val=2)
```

Out[32]:
```
[('_', '-', 12)]
```

How can recent advances in Convolutional Neural Networks and Deep Learning improve the Optical Character Recognition of Persian text?

Introduction

The overall aim of this research is to use deep learning algorithms to improve Optical character recognition (OCR) of the Persian alphabet. OCR can help Persian businesses use document image processing to recognise the contents of documents. More specifically, research is needed to accurately extract Persian numbers and letters and digitise them such that the information contained in the documents can be searched by key words. Currently, if a business that speaks and writes in Persian wants to know the contents of its internal documents, a worker would need to manually input the information in each document one by one. My supervisor Sepehr Jalali et al, 2018 has created a data set that can be used to help to solve this contemporary business problem.

The data set includes business documents written in Persian. These are of different types and topics such as employment documents and business transactions. On each document, there is metadata at the top of the page on the left. There are 3 fields that can be filled in; 'Title' which will be filled in with the title, 'dated' which will be filled in with the date and 'Attachment' which will be filled in if the document is an attachment.

The data set contains 1156 labelled images for all the numbers, 0 – 9, in Persian. Additionally, there are 1156 labelled images for the Persian words for 'Date', 'dated', 'attachment', 'number' and 'title'. The data set has been created such that each image showing the label is in a slightly different font. Then there are 59 images of Persian documents.

There are three main tasks for this project. The first is to be able to use deep learning to accurately recognise where the date is on the page. Then second, once the location is found, to recognise the numbers which represent the dates of the documents and extract this metadata. Once the recognition of numbers has been achieved then, additional tasks include, optical character recognition of letters and words, including the content of the letters. Further work could include the OCR of hand written Persian. Here is a couple of examples of the Persian business letters which are going to be used as test images for the OCR solution being developed:

فرمانده کل قوا
ستاد کل نیروهای مسلح
کمیته جستجوی مفقودین

جمهوری اسلامی ایران
ستاد کل نیروهای مسلح

تاریخ ۹۲/۵۵۲/ک
شماره ۹۲/۲/۱۵۰
پیوست

ل: کمیته جستجوی مفقودین
ا: مسئول محترم مؤسسه فرهنگ دیده‌بان استان فارس
موضوع: ساخت یادمان شهدای گمنام

با درود و صلوات بر محمد و آل محمد و ابراز سلام و تحیت فراوان
با احترام ، بازگشت نامه شماره ۹۱/۱۲/۱۵ مورخ ۹۱/۱۲/۱۵ درخصوص وضعیت نامتعارف قبور شهدای گمنام شهرستان فسا در استان فارس به اطلاع می‌رساند:
ساخت حرم شهدای گمنام هر شهر و شهرستان برعهده مسئولین آن شهر ، بویژه شهرداری و فرمانداری است. لذا از آنان بطور جدی مطالبه شود.

فرمانده کمیته جستجوی مفقودین ستاد کل نیروهای مسلح
سرتیپ پاسدار سید مجتبی [illegible]
از طرف سرتیپ دوم پاسدار ناصر سعیدی

رونوشت:
- اداره حفظ آثار و نشر ارزش‌های دفاع مقدس استان فارس جهت پیگیری.

شماره ۱۰۴۸۰۵
تاریخ ۱۳۹۲/۵/۱۲
پیوست

جمهوری اسلامی ایران
ریاست جمهوری

فقر رئیس جمهور
بازرسی ویژه

جناب آقای دانشجو
رئیس محترم دانشگاه آزاد اسلامی

آفی

با سلام و احترام؛

نظر به اینکه عملکرد آن دانشگاه در این بازرسی تحت بررسی می‌باشد، لطفاً دستور فرمایید موارد زیر را با قید فوریت تهیه و ارسال نمایند.

۱. گزارش عملکرد و صورتهای مالی سال ۹۱ و ۶ ماهه سال جاری
۲. گزارش پرونده مصوب سال‌های ۹۱ و ۹۲
۳. اعلام اسامی اشخاصی که حق گشایش حساب‌های بانکی به نام دانشگاه را دارند
۴. لیست و موجودی حساب‌های سپرده و جاری دانشگاه در بانکها
۵. علت جابجایی حساب‌های بانکی
۶. علت متمرکز کردن وجوه نقد متعلق به سایر واحدها در حساب دانشگاه مرکزی
۷. علت راکد نگه داشتن حساب سپرده‌های متمرکز شده
۸. علت عدم تعهد به قرارداد بانک ملی
۹. علت گشایش حساب در بانک پاسارگاد شعبه سعادت آباد
۱۰. اعضای شورای دانشگاه و کمسیون‌های آموزشی، پژوهشی و دانشجویی
۱۱. آیین نامه مدیریت دانشگاه
۱۲. علت متوقف کردن پروژه‌های عمرانی
۱۳. علت بکارگیری افراد شاغل در سایر دستگاهها در جایگاه‌های رئیس و معاون دانشگاه آزاد
۱۴. علت اخراج و کنارگیری افرادی که از شایستگی و تخصص بالایی برخوردار بوده‌اند
۱۵. آیا سال سابقه و رزومه کاری آقایان محمد غجیت (معاون اداری و مالی دانشگاه)، محمد و مجید مسافرچی و حسین حسین‌پور
۱۶. علت انتصاب آقای نصرو دانشجو به عنوان رئیس واحد علوم و تحقیقات تهران با توجه به اینکه ایشان مشغله کاری زیادی دارند و طبق آیین‌نامه فاقد شرایط لازم برای این پست می‌باشند.

Critical Context

OCR is commonly used and is effective in English and other Latin languages and can be easily accomplished through scanning and photography. However, in Persian and similar languages with related alphabets like Arabic and Urdu, research is still ongoing to create the technology for a similar capability. “There are a large variety of character shapes, sometimes ligature between characters, and an overlapping of the components. Persian words and characters within words are written from right to left.” [1]

Additionally, contemporary OCR models employ auto correct as a final layer, before classification using; “Language models or recognition dictionaries”[2]. However, “using a language model complicates training of OCR systems, and it also narrows the range of texts that an OCR system can be used with.”[2] “Performance on non-linguistic data like codified alphanumerical strings is significantly worse.”[3]

Persian also known as Farsi, is one of the Western Iranian languages within the Indo-Iranian branch of the Indo-European language family. This is in contrast to Arabic which is considered a Central Semitic language [4]. However, Persian uses a modified variant of the Arabic script and therefore its relevant to consider OCR research on Arabic alphabet.

It is interesting to see across the literature a similar assertion of the challenge of OCR for non-Latin scripts and specifically arabic. “(Arabic) is a cursive script…morphologically very rich text…characters are often connected by a baseline thereby forming sub-words. Moreover, one Arabic character may have more than one shape depending on its position in the word. This text has also different texture characteristics compared to Latin or Chinese ones: more stokes in different directions, different font size and aspect ratio, etc. All these particularities make…most (of the) existing OCR systems fail to detect and recognize Arabic Text.”[5]

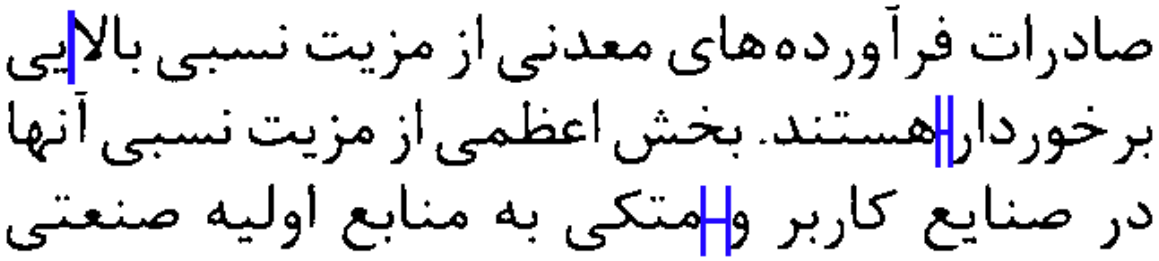
As shown, there are challenges of having an accurate OCR for Arabic alphabet and by extension the Persian alphabet.

Approaches: Methods & Tools for Design, Analysis & Evaluation

In the existing literature, many difference approaches and tools have been used. I looked at OCR literature in general and more specifically at OCR done on Arabic, and Urdu scripts. The current OCR research on the Persian alphabet is very limited, so I was looking to see what insights could be gleaned from the general literature and non-Latin scripts more specifically.

Pre-processing
One of the first steps mentioned in the literature is the pre-processing steps that are used. For this project, it is useful to consider what pre-processing steps would be useful for the recognition of the Persian alphabet.

Segmentation
A debate that occurs in the literature is whether to use a pre-processing step called segmentation. Segmentation allows the separation of lines, words and characters, to consider each segment individually in order to make it easier to classify what word or character it is.



One word segmentation method works by finding spacing between words, See above, an example of Persian[6]. Yet, sometimes there isn't space that can be detected easily or even overlapping occurs between words and letters. [7] According to Yousfi, et al "Although it is intuitive, segmentation-based recognition is an error-prone specially for cursive script with overlapping and ligature cases [8] as well as for texts with complex background." [9]

Similarly, Jain et al writes, "such models did not scale well. In recent years, there has been a shift towards segmentation-free text recognition models, mostly based on Hidden Markov Models (HMM) or Recurrent Neural Networks (RNN)."[10] This shift towards a non-segmentation approach was a key finding of note.

Urdu approaches
Research has also recently been undertaken of using OCR on Urdu. Urdu "is
a Persianised and standardised register of the Hindustani language. It is the official national language and *lingua franca* of Pakistan."[11] Jain et al, created a "model following a segmentation-free, sequence-to sequence transcription approach. The network transcribes a sequence of convolutional features from an input image to a sequence of target labels. This discards the need to segment the input image into its constituent characters/glyphs, which is often arduous for scripts like Urdu."[12]

Document Layout Analysis
One possible approach by Hamdy et al, "uses a deep convolutional neural network (D-CNN) to classify zones/patches of the document image as text or non-text then the text lines and words of textual zones are extracted."[13]

An area of research that has been developing is video OCR. Yousfi, et al. [9] proposed a similar recognition approach to Hamdy et al that processed a "text image without any pre-processing operations or prior segmentation and implemented three methods for Arabic text recognition in the underlying videos. Each approach uses a deep learned model to represent the text image as a sequence of learned features. The experimental results showed that the proposed approach achieves good recognition rates that outperform the existing well-known commercial OCR systems."

Moysset et al used one network, "that detects the left sides of each text lines by predicting the value of the object position coordinates as a regression problem The latter network recognizes the text and predicts the end of the text of the line." [14]

Liao et al, designed a state of the art Text detector for use in natural scenes. It is called Text Boxes, "which is based on fully- convolutional network (LeCun et al. 1998). TextBoxes directly outputs the coordinates of word bounding boxes at multiple network layers by jointly predicting text presence and coordinate offsets to *default boxes* (Liu et al. 2016)."[15]. More formally this text detector is combined with a successful text recognition algorithm, CRNN, to achieve start of the art performance.

I'm interested to investigate whether this CRNN or indeed other learning architectures can be used to transfer learn from my specific data set in Persian or whether it would be better to train a model from scratch. Additionally, I hope to try out this text detector and compare it to existing text recognition methods to identify the regions in the documents. For example, to recognise the 'Title' region. These are all examples of the word 'title' in Persian that are in the data set being used.

Architectures
I intend to try out different architectures and to see what works best. There are different state of the art architectures that have been tried
The expectation from the literature is that CNNs and similar deep learning models will offer the best accuracy. Per Bluche et al, "In HMMs, the characters are modelled as sequences of hidden states, associated with an emission probability model. Gaussian Mixture Models (GMMs) is the standard optical model in HMMs. However, in the last decade, emission probability models based on artificial neural networks have regained considerable interest in the community, mainly due to the deep learning trend in computer vision and speech recognition. In this latter domain, major improvements have been observed with the introduction of deep neural networks."[16]

Amer et al use the following architecture to identify fonts in Arabic. Although this project isn't focussing on fonts per se, it is looking to understand words written in different fonts, and so it

is useful to see what architecture is being used to achieve a high accuracy. “The method mainly uses a deep convolutional neural network (D-CNN) to classify zones/patches of the document image as text or non-text then the text lines and words of textual zones are extracted.”[13] Whether a complex architecture like this is needed, will be something to investigate.

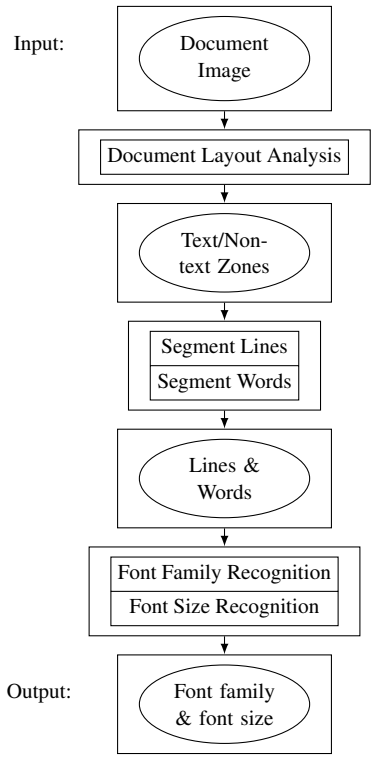
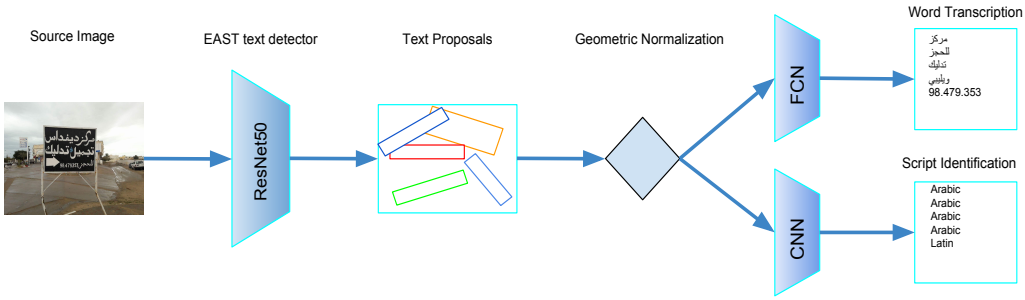


Fig. 1: Proposed System Architecture

Multiple Languages

Some papers are dealing with multiple languages and scripts and are relevant for this project as some of the dates are written in english numbers. Ukil et al, use “a two- and three-layered CNNs for three different scales of the input image; and secondly, we use exactly same CNNs for two different scales of the transformed image (wavelet transform). We then merge those features and make ready for script identification.”

Patel et al, “demonstrate that an existing detection framework when combined with our script identification method achieves state-of-the-art performance on joint detection and script identification.” They describe how their, “OCR does not require any language or script specific information. To (their) knowledge, (they) are first to present a unified OCR for multiple-languages.”[17] Each query text is assigned a script class – see below.[17]



Similarly Breuel et al use LSTMs for this challenge of multi languages. [2]

Ethical concerns

The data set was given to me by my supervisor and I don't foresee any ethical concerns undertaking this work. If additional data sets are used, then copyright issues and permissions would need to be investigated. However definitely for the first task of recognising numbers and dates, my expectation is that I have enough data.

Resources

I may need to use a GPU server; I also need to work on upskilling in using Tensor Flow and Keras.

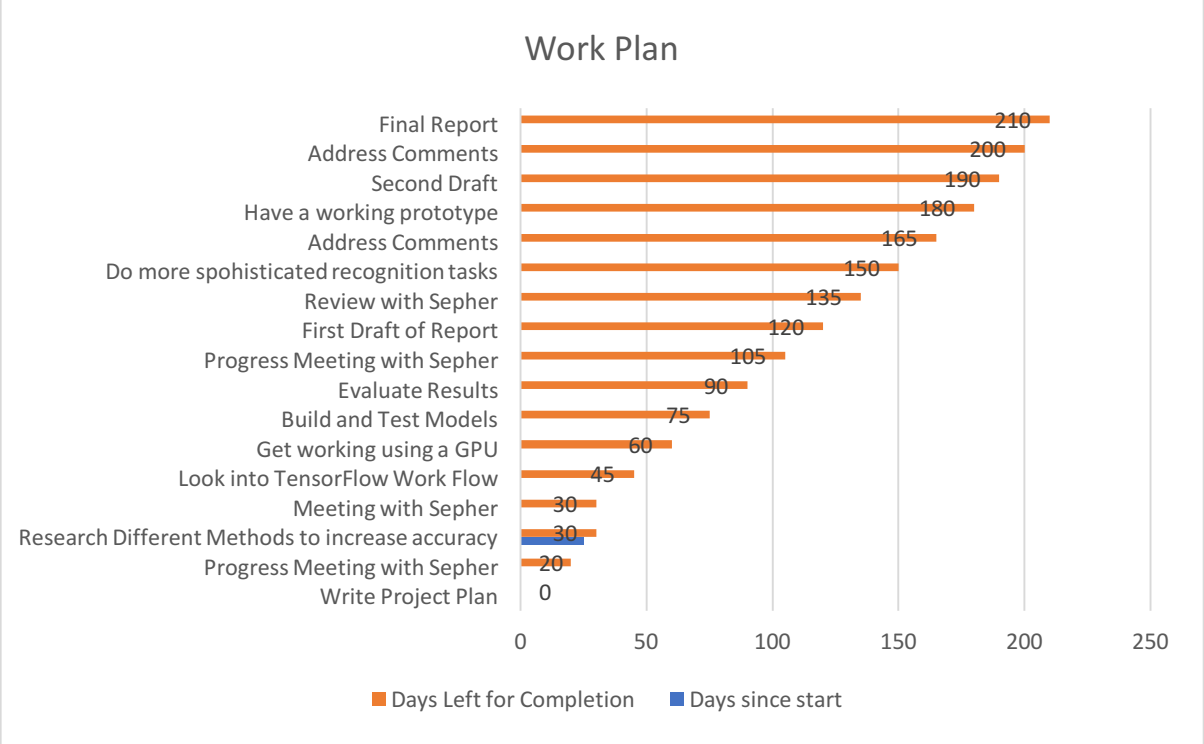
Evaluation

The only current offerings of OCR for Persian I found was by Tesseract and ABBYY. According to ABBYY, "the OCR Farsi feature is only a technical functionality preview. ABBYY's software engineers continue to optimize the code of algorithms for Farsi recognition."[2] As part of the evaluation, I will try these OCR solutions and compare and contrast the accuracy and results with the solution created during this project.

Risk Register

Description	Likelihood (1-3)	Consequence (1-5)	Impact (L x C)	Mitigation
Models are hard to write	3	4	12	Look for example scripts, such as on Fast Ai, Tensor Flow examples
It takes too long to train models	1	4	4	Can use an online server like paperspace.com
City university server not working	2	1	2	Can use an online server like paperspace.com
Loose all my work	1	5	5	I can put it on two Clouds, Dropbox and Google
More data is needed	2	3	6	Can create more data

Work plan



References

[1] MohammedAli Mudhsh and Rolla Almodfer, "Arabic Handwritten Alphanumeric Character Recognition Using Very Deep Neural Network," *Information*, vol. 8, no. 3, p. 105, Aug. 2017.

[2] A. Ul-Hasan and T. M. Breuel, "Can we build language-independent OCR using LSTM networks?," 2013, p. 1.

[3] R. Milan Troller, "Practical OCR system based on state of art neural networks," in *Faculty of Electrical Engineering Department of Cybernetics*, 2017, vol. 2, pp. 629–633.

[4] "The Persian Language." .

[5] C. Sonia Yousfi, "Embedded Arabic text detection and recognition in videos," PhD Thesis, INSA de Lyon, 2016.

[6] I. Vynckier, "Segmenting Words and Characters | How OCR Works," *How OCR Works | A Close Look at Optical Character Recognition*. [Online]. Available: http://www.how-ocr-works.com/OCR/word-character-segmentation.html. [Accessed: 30-Apr-2018].

[7] L. S. Alhomed and K. M. Jambi, "A Survey on the Existing Arabic Optical Character Recognition and Future Trends."

[8] J. H. Alkhateeb, "Off-Line Arabic Handwritten Isolated Character Recognition," *Int. J. Eng. Sci. Technol.*, vol. 7, no. 7, p. 251, 2015.

[9] S. Yousfi, S.-A. Berrani, and C. Garcia, "Deep learning and recurrent connectionist-based approaches for Arabic text recognition in videos," in *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, 2015, pp. 1026–1030.

[10] M. Jain, M. Mathew, and C. V. Jawahar, "Unconstrained scene text and video text recognition for Arabic script," in *Arabic Script Analysis and Recognition (ASAR), 2017 1st International Workshop on*, 2017, pp. 26–30.

[11] "Urdu," *Wikipedia*. 29-Apr-2018.

[12] M. Jain, M. Mathew, and C. V. Jawahar, "Unconstrained OCR for Urdu using Deep CNN-RNN Hybrid Networks," 2017.

[13] Salma Hamdy, Ibrahim M. Amer, and Mostafa G. M. Mostafa, "Deep Arabic Font Family and Font Size Recognition." International Journal of Computer Applications (0975 - 8887), Oct-2017.

[14] B. Moysset, C. Kermorvant, and C. Wolf, "Full-Page Text Recognition: Learning Where to Start and When to Stop," *ArXiv Prepr. ArXiv170408628*, 2017.

[15] M. Liao, B. Shi, X. Bai, X. Wang, and W. Liu, "TextBoxes: A Fast Text Detector with a Single Deep Neural Network," p. 7.

[16] T. Bluche, H. Ney, and C. Kermorvant, "A comparison of sequence-trained deep neural networks and recurrent neural networks optical modeling for handwriting recognition," in *International Conference on Statistical Language and Speech Processing*, 2014, pp. 199–210.

[17] Y. Patel, M. Bušta, and J. Matas, "E2E-MLT-an Unconstrained End-to-End Method for Multi-Language Scene Text," *ArXiv Prepr. ArXiv180109919*, 2018.

[18] "OCR for Farsi [Technology Portal]." [Online]. Available: https://abbyy.technology/en:features:ocr:farsi-ocr. [Accessed: 30-Apr-2018].

Ethics Form

A.1 If your answer to any of the following questions (1 – 3) is YES, you must apply to an appropriate external ethics committee for approval.		*Delete as appropriate*
1.	Does your project require approval from the National Research Ethics Service (NRES)? For example, because you are recruiting current NHS patients or staff? If you are unsure, please check at http://www.hra.nhs.uk/research-community/before-you-apply/determine-which-review-body-approvals-are-required/.	**No**
2.	Does your project involve participants who are covered by the Mental Capacity Act? If so, you will need approval from an external ethics committee such as NRES or the Social Care Research Ethics Committee http://www.scie.org.uk/research/ethics-committee/.	**No**
3.	Does your project involve participants who are currently under the auspices of the Criminal Justice System? For example, but not limited to, people on remand, prisoners and those on probation? If so, you will need approval from the ethics approval system of the National Offender Management Service.	**No**
A.2 If your answer to any of the following questions (4 – 11) is YES, you must apply to the City University Senate Research Ethics Committee (SREC) for approval (unless you are applying to an external ethics committee).		*Delete as appropriate*
4.	Does your project involve participants who are unable to give informed consent? For example, but not limited to, people who may have a degree of learning disability or mental health problem, that means they are unable to make an informed decision on their own behalf?	**No**
5.	Is there a risk that your project might lead to disclosures from participants concerning their involvement in illegal activities?	**No**
6.	Is there a risk that obscene and or illegal material may need to be accessed for your project (including online content and other material)?	**No**
7.	Does your project involve participants disclosing information about sensitive subjects? For example, but not limited to, health status, sexual behaviour, political behaviour, domestic violence.	**No**
8.	Does your project involve you travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning? (See http://www.fco.gov.uk/en/)	**No**
9.	Does your project involve physically invasive or intrusive procedures? For example, these may include, but are not limited to, electrical stimulation, heat, cold or bruising.	**No**
10.	Does your project involve animals?	**No**
11.	Does your project involve the administration of drugs, placebos or other substances to study participants?	**No**

A.3 If your answer to any of the following questions (12 – 18) is YES, you must submit a full application to the Computer Science Research Ethics Committee (CSREC) for approval (unless you are applying to an external ethics committee or the Senate Research Ethics Committee). Your application may be referred to the Senate Research Ethics Committee.		*Delete as appropriate*
12.	Does your project involve participants who are under the age of 18?	**No**
13.	Does your project involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)? This includes adults with cognitive and / or learning disabilities, adults with physical disabilities and older people.	**No**
14.	Does your project involve participants who are recruited because they are staff or students of City University London? For example, students studying on a specific course or module. (If yes, approval is also required from the Head of Department or Programme Director.)	**No**
15.	Does your project involve intentional deception of participants?	**No**
16.	Does your project involve participants taking part without their informed consent?	**No**
17.	Does your project pose a risk to participants or other individuals greater than that in normal working life?	**No**
18.	Does your project pose a risk to you, the researcher, greater than that in normal working life?	**No**

A.4 If your answer to the following question (19) is YES and your answer to all questions 1 – 18 is NO, you must complete part B of this form.		
19.	Does your project involve human participants or their identifiable personal data? For example, as interviewees, respondents to a survey or participants in testing.	**No**