# Machine Learning Engineer Nanodegree

## Capstone Project

Alex Fong Jie Wen
December 11th, 2017

## Project Overview

This project tests the feasibility of convolutional neural networks (CNNs) for Japanese OCR, on mobile devices. Feature map pruning is used to reduce the size of the model & increase inference speed. The first part of the report discusses the domain background and related works. The second part covers analysis of the ETL2 & ETL9G Japanese character datasets, CNNs to be attempted and detailed steps on CNN pruning. The performance of CNNs and effectiveness of pruning are explored in the third part of the report.

## 1. Definition

### Domain Background

Japanese consists of four types of scripts, kanji, hiragana, katakana and roman numerals. Symbols are also commonly used. Offline Japanese character recognition has been applied to a number of applications, such as transcription of documents and even as a tool for learning the language.

Convolutional neural networks (CNNs) have been used to obtain state-of-the-art performance in character recognition for various languages [1-3] but few attempts were made for the Japanese language.

There is also remarkable progress in research for using CNNs in portable devices. However, the focus has been on commonly used languages such as Chinese [2], or CNNs in general [5].

### Problem Statement

This capstone project aims to explore the feasibility of conducting deep learning based optical character recognition (OCR) for printed Japanese text, on mobile devices. To focus the scope of the project, handwritten text will not be considered in testing. This is fair, given that OCR is used more frequently for printed material rather than written material with the exception of mail sorting and other niche usages.

### Solution Statement

We desire a model which runs without requiring large amounts of memory, and makes predictions at an acceptable speed on mobile phone CPUs. A good estimate is below 100mb of memory, and at least 5 characters in 1 second. Japanese and Chinese share a large number of characters. Human level accuracy for handwritten Chinese is 96.1% [6]. A similar or higher recognition rate is desired for printed Japanese text.

## Dataset and Inputs

The ETL2 and ETL9-G datasets will be used for this project. The datasets in the ETL character database have been collected by Electrotechnical Laboratory, universities and other research organizations for character recognition researches from 1973 to 1984.

ETL2 contains images of 2304 different Japanese characters printed for newspapers and patent applications. Each character is printed the same number of times by printing press machines. The Japanese characters consist of JIS level one Kanji, hiragana, katakana, the roman alphabet and symbols. The characters come in 2 different fonts, Mincho and Gothic. Each image has dimensions 60 by 60. There are 52769 images in total.

ETL9-G contains images of 3036 different Japanese characters, each character written the same number of times by 4000 different writers. JIS level one Kanji, hiragana and katakana, are present in this dataset. Each image has dimensions 128 by 127. There are 607200 images in total.
Both the ETL2 & ETL9-G datasets are balanced due to the use of datasheets for data collection.

Datasheets can be found here:
http://etlcdb.db.aist.go.jp/etlcdb/etln/etl2/e2code.jpg
http://etlcdb.db.aist.go.jp/etlcdb/etln/etl9/e9sht.htm

Dataset details can be found here:
http://etlcdb.db.aist.go.jp/?page_id=1721
http://etlcdb.db.aist.go.jp/?page_id=1711

## Evaluation Metrics

The models produced will be evaluated based on precision, amount of storage required, and the average time taken for each character on a single threaded CPU or possibly a mobile CPU. Runtimes and memory usage will be recorded in python.

Accuracy, also known as recall, will also be used as a measure of evaluation, for comparison with the human recognition rate benchmark. F1 score is not as important in tasks such as OCR. However, it will be measured as well.

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\mathrm{recall}} + \frac{1}{\mathrm{precision}}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

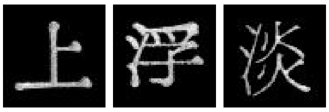
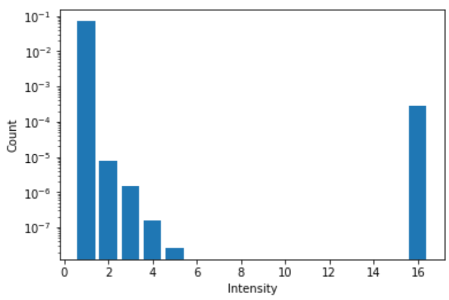
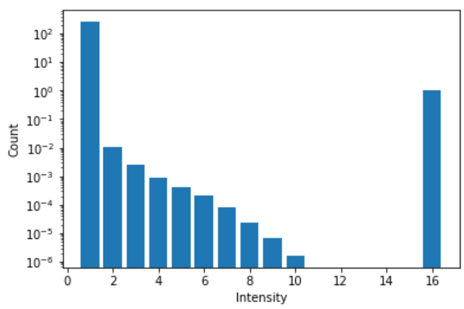
$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

where tp = true positives, fp = false positives, fn = false negatives

## II. Analysis

### Data exploration

| Dataset: | ETL2 | ETL9-G |
|---|---|---|
| Samples: | <br>- characters fill most of the image<br>- resembles target input<br><br>(brightness increased by 4x) | <br>- characters do not fill the whole image<br>- noisy at the edges, might contain small portion of another word<br>- resembles target input after affine transformation/warping<br><br>(brightness increased by 16x) |
| Bit depth: | 6 | 4 |
| Mean: | 29.907 | 11.317 |
| $\sqrt{Var}$ : | 65.511 | 39.100 |
| Log histogram of mean image:<br>(refer to show_histograms.ipynb for code) |  |  |

As seen from the above comparisons, the ETL2 and ETL9-G datasets are extremely different from each other.

The first sign is a 2x difference in mean and variance.
The second is the difference in histogram shape for the mean image. ETL9-G's higher kanji count should give fewer values at 0, more values at 16, and retain a similar shape if the two datasets were similar. Such was not observed. The magnitude of counts for ETL9-G images are also on a different scale as compared to ETL2.

ETL2 & ETL9-G possess a different distribution of pixel intensities. The two datasets should and will be handled separately during data preprocessing.

### Benchmark Model

A good benchmark model for memory and computation time is the results obtained by Xiao et al [2] for Chinese OCR. Chinese characters are extremely similar to Japanese characters, partially sharing the same character set. Their network required 2.3MB of storage and took 9.7 ms per character image on a single threaded CPU. The network had a top-1 error of 2.91%.

They have utilized feature map pruning, loop unrolling, and low rank factorization to achieve these results. Feature map pruning with sparse matrix operations resulted in a 2.5x improvement in computational speed, and 10x less memory required. Only feature map pruning will be used in this project. A similar improvement in computation time and memory use is desired.

A good benchmark model for accuracy is the results obtained by Tsai [4]. He mentions that his network for handwritten Japanese character recognition had an estimated recognition rate of above 96.1%, the accuracy rate for human recognition.

## Algorithms and Techniques

Quick overview of CNNs:
CNNs consist of many layers. Commonly used layers are convolutional (conv) layers, pooling layers, batch normalization, activation function layers and fully connected layers. These layers are stacked together to convert input, such as images, into usable output, such as values for predicting probabilities.

CNNs are trained by first computing a loss value using a lost function, and shifting the trainable parameters to minimize the loss. The gradient of loss differentiated w.r.t to trainable parameters is calculated through a process called back-propagation. These gradients are used to adjust parameters and thus minimize loss.

Conv layers pass a trainable filter throughout the input, generating an output vector through convolution. Parameters such as padding, filter size and stride modify output dimensions and properties. The filters pick out features such as gradients and patterns related to the input, and the conv layer activates when the features are present.

Activation function layers like the ReLU exist to threshold outputs by conv layers. Some layers, such as the PRelu, contain trainable parameters.

Pooling layers reduce the size of input, controlling overfitting and reducing the amount of parameters and computation in the network. A predetermined filter is passed through the input to achieve the above effect. For instance, a max pool layer picks out the largest item in the filter and returns that as output.

Batch normalization layers assist in training neural networks, circumventing the problem of vanishing gradients during back-propagation.

Fully connected layers are similar to conv layers, but every element of the input vector is connected to each other in generating output. They are generally used for computing classification scores.

Transfer learning is the improvement of learning a new task through transfer of knowledge from a related task that has already been learned. Many popular CNNs have been trained on imagenet, an extremely large dataset for classification of living things and objects. While OCR seems to be quite distant from living things and objects, we might still be able to take advantage of transfer learning.

VGG pretrained on imagenet will be fine-tuned and tested on our datasets, as an attempt at transfer learning. Custom networks proven effective for OCR in other languages will also be tested.
The models will be trained with classification cross entropy loss, stochastic gradient descent (SGD) and stepped learning rate.

Models:

VGG11 with batch normalization (VGG11_BN)

Architecture:
Input-64C3-MP3-128C3-MP3-256C3-256C3-MP3-512C3-512C3-MP3-4096FC-4096FC-Output.
Each conv layer has n outputs (nC3) and is followed by a BN layer & ReLU.
The conv layers have kernel size of 3x3, padding of 1 and stride of 1.
The max pooling layers (MP3) have kernel sizse of 2x2 and stride of 2.
The first 2 fully connected layers (FC) are followed by a ReLU and dropout layer.
The output layer is a fully connected layer with output dimension equal to the number of classes.

Neural network for Chinese OCR (CNet)

Architecture:
Input-96C3-MP3-128C3-MP3-160C3-MP3-256C3-256C3-MP3-384C3-384C3-MP3-1024FC-Output.
Each conv layer has n outputs (nC3) and is followed by a BN layer & PReLU.
The conv layers have kernel size of 3x3, padding of 1 and stride of 1.
The max pooling layers (MP3) have kernel size of 2x2 and stride of 2.
The fully connected layer (FC) is followed by a PReLU and dropout layer.
The output layer is a fully connected layer with output dimension equal to the number of classes.

Feature map pruning will be conducted on the best performing networks to improve inference speed and reduce memory consumption.

Pruning will be conducted with the approach suggested by Molchanov et al. [5]. They have proven that feature map pruning using taylor estimation provides the best results, and each pruning iteration should be followed by fine-tuning to retain inference accuracy.

Only feature maps from conv layers will be pruned in this project. This is acceptable given conv layers make up bulk of CNNs.

# III. Methodology

## Data Preprocessing

Both datasets are normalized separately given their differences in mean, variance and size.
The models will be trained with the ETL2 set alone for preliminary testing, and later with both the ETL2 + ETL9-G set.

Image augmentation was deemed unnecessary as the ETL9-G set is sufficiently noisy, as determined previously. Using both the ETL2 & ETL9-G set is likely to provide accurate enough results.

The datasets are split into 3 sets using stratified sampling, with the following distributions: train (68%), validation(16%) and test (20%). Stratified sampling was used as each particular word does not have that many occurances. There is a high chance of a class not being represented in the training set with basic sampling.

## Implementation

VGG11_BN was loaded with weights from ImageNet training, and trained with 20 epochs of ETL2. We used SGD with initial learning rate of 0.001, momentum of 0.9. Learning rate is decayed every 7 epochs, by a multiplicative factor of 0.1. Decaying learning rate has been observed to speed up training[7]. Batch size is 32. The model was tested after every epoch. As a form of early stopping, the weights giving the least loss during inference were chosen as the final weights for the model.

The same was conducted on CNet.

The better network of the 2 was trained with both ETL2 and ETL9-G, and subsequently pruned until only 20% of convolutional feature maps remained. 100 iterations of fine-tuning were conducted after pruning each feature map.

The detailed pruning process is as follows:
(see paper by Molchanov, P., et al[5] for detailed derivation and analysis)

Input is passed through the network, and gradients of weights in each conv layer is calculated with respect to the loss. The importance of each feature map is then estimated with the formula below.

$$\Theta_{TE}(z_l^{(k)}) = \left| \frac{1}{M} \sum_m \frac{\delta C}{\delta z_{l,m}^{(k)}} z_{l,m}^{(k)} \right|,$$

$\theta_{TE}$ refers to the importance of the weights of an arbitrary feature map $z_l^{(k)}$.
$z_{l,m}^{(k)}$ refers to an output of the feature map.
M is the length of the feature map, and C the result from the loss function.
$\frac{dC}{d(x)}$ is the gradient of the cost function w.r.t. to the activation.

The importance estimates are then normalized by their magnitude as each layer has different dimensions.

$$\hat{\Theta}(\mathbf{z}_l^{(k)}) = \frac{\Theta(\mathbf{z}_l^{(k)})}{\sqrt{\sum_j \left(\Theta(\mathbf{z}_l^{(j)})\right)^2}}.$$

The feature map with the smallest importance value is removed along with the sections that directly depend on its outputs. Each pruning operation is followed by iterations of fine-tuning.

The following is pseudo code for the pruning process:

```
Repeat for desired number of pruning cycles:

    # pruning
    pass 1 batch of data through the model, storing the feature maps of all layers
    calculate the loss
    calculate the importance of each feature map with equation above

    # finetuning
    train the model on 250 batches of data
```

The actual implementation uses object oriented programming and multiple functions. Refer to src.prune.py and src.nn.models.py.

The number of pruning cycles required are estimated by summing the memory consumption of all conv filters, multiplying by the desired percentage and then dividing by the size of each conv filter. This determines how many conv filters to prune. Each pruning cycle removes one filter.

## Refinement

Experimentation on VGG has shown that transfer learning provides little to no benefits for OCR. Considering its large memory requirement which goes against the goal of a small memory footprint, further experimentation on VGG was halted.

The pruning process was refined to use 250 fine-tuning iterations rather than 100. Molchanov, P., et al[5] observed that more iterations of fine-tuning helps the model retain it's original accuracy. The pruning process was also extended to test the limits of pruning, stopping after 90% of weights were pruned.

| | Before | After | After |
|---|---|---|---|
| Network: | CNet-80 (80% pruned), 100 iterations | CNet-80 (80% pruned), 250 iterations | CNet-90 (90% pruned), 250 iterations |
| F1 Score: | 0.9532 | 0.9863 | 0.9448 |
| Accuracy: | 0.9652 | 0.9890 | 0.9573 |

As observed by Molchanov, P., et al[5], increasing the fine-tuning iterations results in better preservation of accuracy after pruning, with a difference of about 3%.

# IV. Results

## Model Evaluation and Validation

Network performance on “Test” data

| Network: | VGG11_BN (ETL2) | CNet-0 (ETl2) | CNet-0 (ETl2+ETL9-G) | **CNet-80 (80% pruned)** | CNet-90 (90% pruned) |
|---|---|---|---|---|---|
| Size, MB: | 1101.2 | 58.15 | 58.15 (66.25 after adding variables for pruning) | **slightly < 31.22** | slightly < 27.15 |
| F1 Score: | 0 | 0.9879 | 0.9890 | **0.9863** | 0.9448 |
| Accuracy: | 0.060 | 0.9913 | 0.9963 | **0.9890** | 0.9573 |
| Duration per image on PC: | 8.25s | 1.73s | 1.55s | **0.364s** | 0.204s |
| Estimated duration per image on Mobile: | - | - | At least 11.24s | **At least 2.64s** | At least 1.48s |

CNet-80’s performance on “Train”, “Validation” and “Test” data.

| | Train | Validation | Test |
|---|---|---|---|
| F1 Score: | 0.9928 | 0.9860 | **0.9863** |
| Accuracy: | 0.9950 | 0.9887 | **0.9890** |

(refer to benchmarks.ipynb for code & output)

Test hardware:
CPU for inference: Intel i3-3140
GPU for training: GTX 1060 6GB
Mobile CPU: Cortex-A53, a low end CPU from 2015.

Estimating model size:
Model size was estimated by summing the size of parameters in each model.
Each float32 takes 4 bytes. For reference, squeezenet takes about 12.76MB.

Estimating mobile performance:
The Caffe2 library is not mature enough for benchmarking CNet on Android. Estimates will be made to predict mobile performance. Squeezenet takes about 0.566s on the Cortex-A53, and 0.0781s on the

Intel i3. Assuming at best linear scaling in performance, the desktop CPU performs 7.25x faster than the mobile CPU. This factor is used to estimate performance on mobile.

The above networks have all been tested on unseen data in the “test” set of the data split. The accuracy score obtained is representative of each model’s performance on unseen data.

The low accuracy score by VGG11_BN shows that Imagenet is too dissimilar to OCR for transfer learning to be successful. While it was possible to adjust the learning rate to more strongly adjust weights, the large memory consumption and long inference time made further experimentation on VGG11_BN undesirable.

Accuracy:
CNet-0 performed extremely well, with accuracy of 0.996, outperforming our benchmark of 0.961. This performance carries on for CNet-80 with accuracy of 0.989. CNet-90 suffers from a great loss in accuracy in comparison to the CNet-80, showing the upper limit of feature map pruning.

Size:
CNet-0 itself is smaller than 100 MB even before pruning, already fulfilling the requirement for memory consumption. Unfortunately, our pruned CNets were unable to match up with the pruning by Xiao et al [2]. Only a 2x reduction in memory consumption was achieved by pruning.

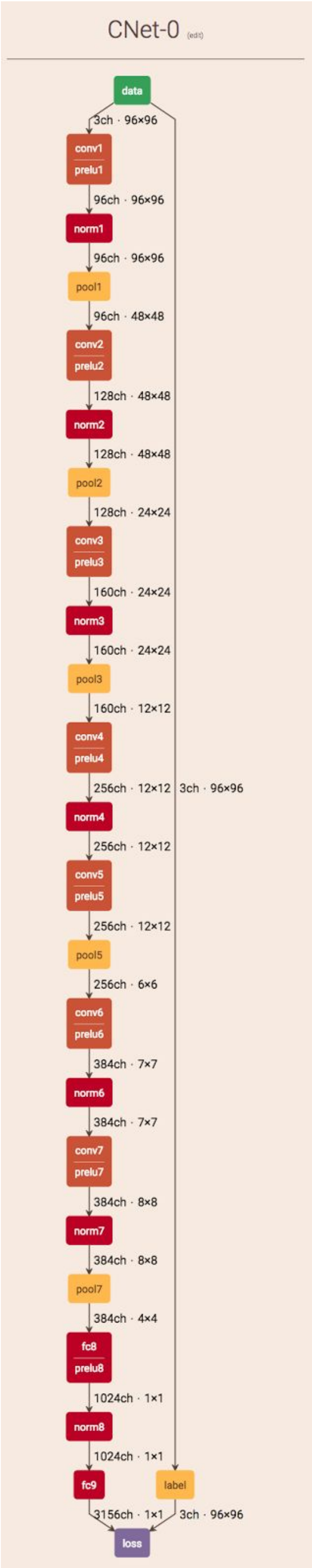
Speed:
The pruned CNets meet the expected speed gain of pruning by Xiao et al [2], with at 4x & higher improvement in speed after pruning as compared to 2.5x on their end. CNet-90 is almost 2x faster than the CNet-80. However the 4% drop in accuracy makes it an infeasible choice. The inference speed of CNet-80 fails to reach the speed benchmark. Pruning CNet is not sufficient for deployment on lower middle end mobile CPUs.

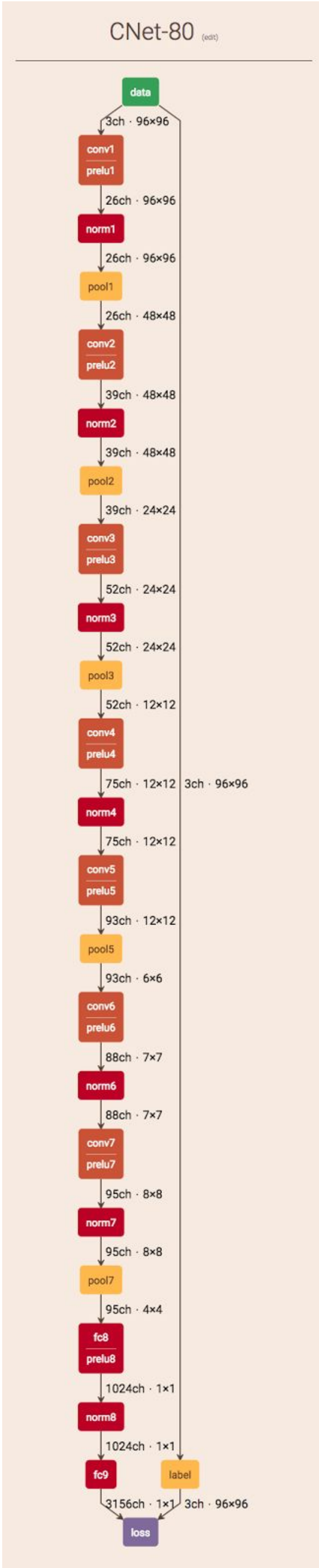
## Justification

Initial estimates that pruning alone is sufficient stem from publicly available benchmarks made on more powerful and recent mobile processors. CNNs run 5x faster on the top end devices such as the Samsung S7 and Google Pixel. Pruning is unable to fill this large gap in performance. Complementary techniques such as loop unrolling and low rank factorization of weights as attempted by Xiao et al [2] are required to obtain desired performance on lower middle end mobile devices.

The lower than expected decrease in network size after pruning is likely due to skipping the pruning process for the fully connected layer. Our assumption on fully connected layers being negligible was not valid. This could be attributed to the large number of classes in Japanese OCR (more than 3000). The fully connected layer for output ended up taking a large amount of space in CNet.

# V. Conclusion




### Pruning Visualization

To the left are visualizations of CNet-0 and CNet-80. As observed, more than half the outputs were dropped from all convolutional layers. This gets more extreme for CNet-90, where the first convolutional layer has only 16 outputs as compared to 96 before pruning.

The drop in convolutional layers results in fewer operations required, speeding up inference.

### Reflection

Implementing the Taylor Expansion method for pruning convolutional layers was challenging but rewarding. Doing so required strong understanding of the source paper, and a well formulated workplan where time is set aside for testing at various stages of development.

Dynamic computational graphs in Pytorch allowed the use of object oriented programming for clear code organization, making the coding process extremely smooth.

Unfortunately, the same success was not found with Caffe2, a companion to Pytorch for deployment of mobile networks. Even so, rough estimations are still possible.

CNet-80 does not fit expectations for the problem. However, there are still many takeaways from this project. It is clear that transfer-learning does not always work as expected, especially when the target domain is too different from the training domain. Simple networks perform very well for OCR, and there may be room for research on even smaller and simpler networks.

## Improvement

This project has many areas for improvement, namely the following:

- prune fully connected layers
- attempting low rank factorization
- trying more exotic network architectures, such as squeezenet.

Another logical extension is gathering data from more diverse sources, such digitized books and their physical counterparts. Testing on such data would shed light on real world performance, especially when algorithms for image segmentation and preprocessing do not work as desired.

Finally, performing live tests on a physical device rather than making estimations would also help with evaluating real word performance.

# Appendix

See src.nn.models for implementation of models
See src.nn.prunable_nn for implementation of prunable Conv2d, PreLU, ReLU, BatchNorm, FC
See src.prune for training code
See src.prune for pruning code

# References


[1] Zhang, X., Bengio, Y., & Liu, C. (2016, June 18). Online and Offline Handwritten Chinese Character Recognition: A Comprehensive Study and New Benchmark. Retrieved December 10, 2017, from https://arxiv.org/abs/1606.05763

[2] Xiao, X., Jin, L., Yang, Y., Yang, W., Sun, J., & Chang, T. (2017, February 26). Building Fast and Compact Convolutional Neural Networks for Offline Handwritten Chinese Character Recognition. Retrieved December 10, 2017, from https://arxiv.org/abs/1702.07975

[3] Wojna, Z., Gorban, A., Lee, D., Murphy, K., Yu, Q., Li, Y., & Ibarz, J. (2017, August 20). Attention-based Extraction of Structured Information from Street View Imagery. Retrieved December 10, 2017, from https://arxiv.org/abs/1704.03549

[4] Tsai, C. Recognizing Handwritten Japanese Characters Using Deep Convolutional Neural Networks Retrieved December 10, 2017, from https:*//cs231n.stanford.edu/reports/2016/pdfs/262_Report.pdf*

[5] Molchanov, P., Tyree, S., Karras, T., Aila, T., & Kautz, J. (2017, June 08). Pruning Convolutional Neural Networks for Resource Efficient Inference. Retrieved December 10, 2017, from https://arxiv.org/abs/1611.06440

[6] F. Yin, Q.-F. Wang, X.-Y. Zhang, and C.-L. Liu. ICDAR 2013 Chinese Handwriting Recognition Competition. 2013 12th International Conference on Document Analysis and Recognition (ICDAR), pages 1464–1470, 2013.

[7] "Transfer Learning - CS231n Convolutional Neural Networks for Visual ...." http://cs231n.github.io/transfer-learning/. Accessed 7 Nov. 2017.