Neural Networks: Pooling Strategies

Jacob Crabtree
Anthony Cavallero
Jason Clemens

Introduction:

Deep convolutional neural networks have been shown to be useful when dealing with data with a gridlike structure. Their most common application is image processing, since neighboring pixels in an image contain related information.

In general terms, convolution is a process of applying one function to another. We denote the convolution of functions f and g as $f * g$. The resulting function of convolution is $(f * g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)\,dt$. In more intuitive terms, we are taking the integral of an input function multiplied by a kernel function, and "sliding" the kernel function forwards by a factor of x. In neural networks, convolution behaves similarly, but has minor differences because we are dealing with a finite discrete set of values instead of the infinite continuous values seen in the general case.

Convolutional layers in neural networks have a single matrix called the kernel which contains a set of parameters. This kernel is convolved across the input layer by "sliding" the kernel across the input at every possible offset. At each position, the output is found by calculating the dot product of the input and kernel. Convolutional layers typically replace dense layers, and are useful because they require fewer learnable parameters and have sparse connectivity, both of which can improve the efficiency of a neural network. However, the dot product lies at the core of convolution, so it is a linear operation. Non-linearity is a requirement of many neural networks because some problems cannot be solved linearly.

Pooling is an approach commonly used in conjunction with convolutional layers in neural networks to summarize an area of its input. Pooling is similar to convolution in that each output is only dependent on a small number of neighboring inputs, but it is different because it allows the use of an arbitrary commutative function rather than dot product. The most common pooling methods are max pooling, which outputs the maximum input value in an area, and average or sum pooling, which output the sum of all their inputs, optionally dividing by the number of inputs.

Max pooling adds nonlinearity to a model, which can be useful for complicated classification, and both types of pooling cause a loss of information. Pooling is used to achieve invariance to input transformations, compact representations, and better robustness to small changes in inputs. [5]

Hypothesis:

In *A Theoretical Analysis of Feature Pooling in Visual Recognition*, Boureau et al. predict that max pooling is better suited for sparse data than average pooling. We hypothesize that this is the case, because with average pooling, a large number of inputs close to zero will easily weaken the influence of a single strong input, but max pooling will only let the strong input through, better representing the existence of features in the input. We further predict that average pooling may perform better than max pooling in scenarios with dense data, because average pooling loses less information in a situation where more information is probably useful for image classification. Finally, we hypothesize that all types of pooling can act as a form of regularization, so we expect to see better validation accuracy when using pooling compared to models without pooling layers.

Experimental Setup:

Before embarking on the actual testing of our hypothesis, there were some basic items that needed to be handled. First and foremost, how would we actually design the learning algorithms. Tensorflow is always a strong option, however, it is rather unwieldy in terms of rapid development and testing. Thus an alternative to the usage of Tensorflow was leveraging the high level API of Keras[1]. The Keras API performs abstractions from the Tensorflow backend, and allows us to surmise many lines of code for layers such as Pooling and Activation into simple 1 line calls to the API. This enabled the group to build, test, and modify models quickly and easily which pushed the bottleneck of development time to training the models.

Another important item was gathering the data in order to train our algorithms. For the sake of simplicity in the short timeframe we had to actually train and develop the models, two popular small image dataset were chosen since training time and image sizes are proportional. The first of the small image datasets was CIFAR-10[2]. The CIFAR dataset contains 60,000 color images of size 32x32 pixels across 10 different classes of wildlife and modes of transportation like dogs, cats, cars, and trucks. These images can be categorized as dense due to the color distributions and background objects that are not the target being included. The other image dataset was MNIST[4]. Similarly to CIFAR, our MNIST data contains 60,000 images, but of size 28x28 pixels, black and white instead of color, and of handwritten digits 0-9.

Finally, the only item left to address in the setup portion was the tools used to train the algorithms. Initially, training was to be conducted on personal computers with GPU boosting from Nvidia GeForce GTX 1080 and 1050ti cards, however the training time was not ideal. While searching for alternatives, Google's Colaboratory[7] quickly rose as a front runner. Colaboratory is a free research space for all to use, and provides both Google Drive and Github support for interactive python notebooks. The python code is loaded into the notebook, like our Keras models, and then run in the cloud with both GPU and TPU acceleration. For the purpose of our work, the GPU acceleration provided by a Nvidia Tesla k80 provided by the Colaboratory servers cut training time from days down to mere hours.

Methodology:

To test our hypothesis across the 2 image databases we needed 2 models, one for Average Pooling and one for Max Pooling, were needed for the datasets. The models made were almost identical, all shared the general constructions of an input layer followed by 3 convolutional, pooling, activation layers and an output layer. The only major difference was whether the models used Average or Max Pooling. In addition to the 4 models that were developed, each model was developed to iterate through a few different hyperparameters in a grid like fashion. In terms of the pooling kernels, or how many pixels are included in the average or max pooling function, the values were 2x2, 3x3, 4x4, 8x8, and for a full quadrant of the image 16x16 was used in CIFAR and 14x14 for MNIST. In addition to the pool sizes we also looked at pooling strides, or the pixel gap between applications of our convolution operation on the data. The CIFAR models were trained for 100 epochs, which allowed for convergence in accuracy, on Colaboratory while the MNIST models only needed to be trained for 10 epochs to reach convergence. Once training and testing of each model was completed, the model was saved for later usage along with the training, validation, and test accuracies for analysis and figure generation.

Results:

Figure 1 shows us how when we fix the kernel size and vary the stride for our models what impact these changes had on the accuracy.

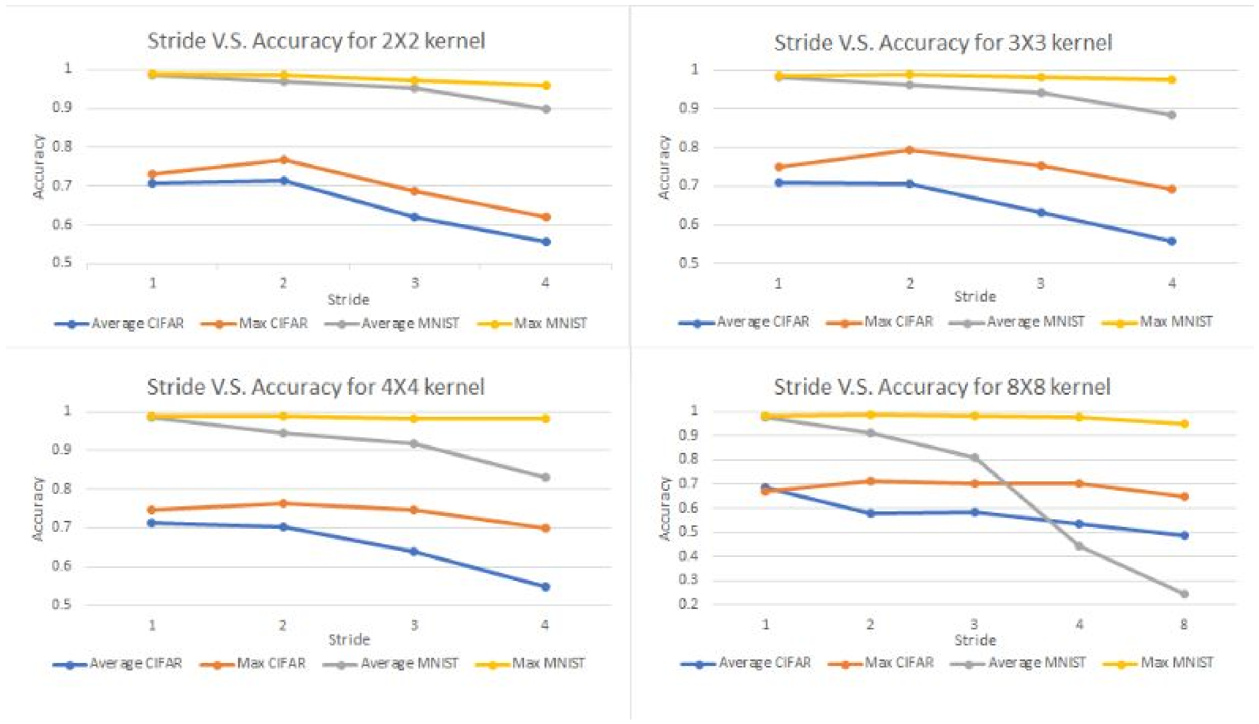


Figure 1: Effects of Stride

Figure 2 shows the same type of information as Figure 1, however because MNIST and CIFAR have different final kernel sizes they could not be compared in the same manner as the previous figure.

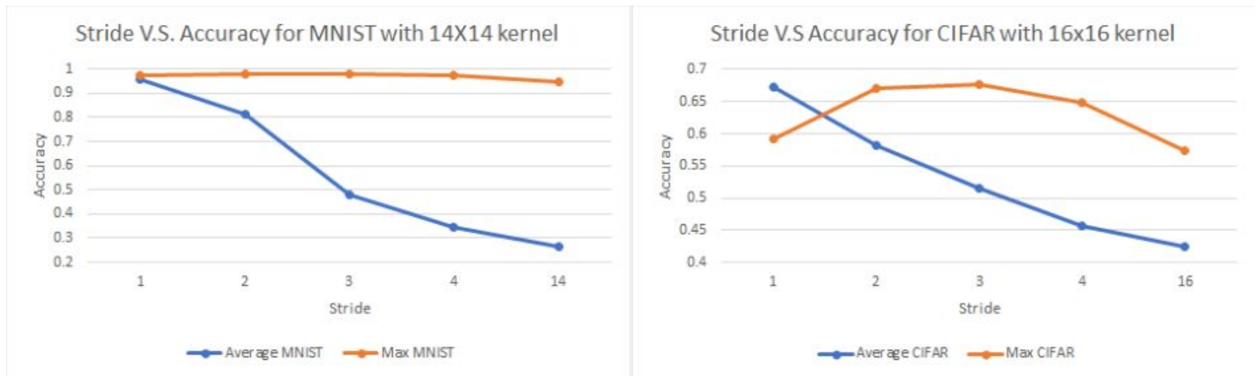


Figure 2: Effects of Stride on Larger Pools

Figure 3 shows us how when we fix our stride and vary our kernel sizes impact the accuracy of the CIFAR models since not every model was trained on strides higher than 4, we only show the all models up to stride 4.

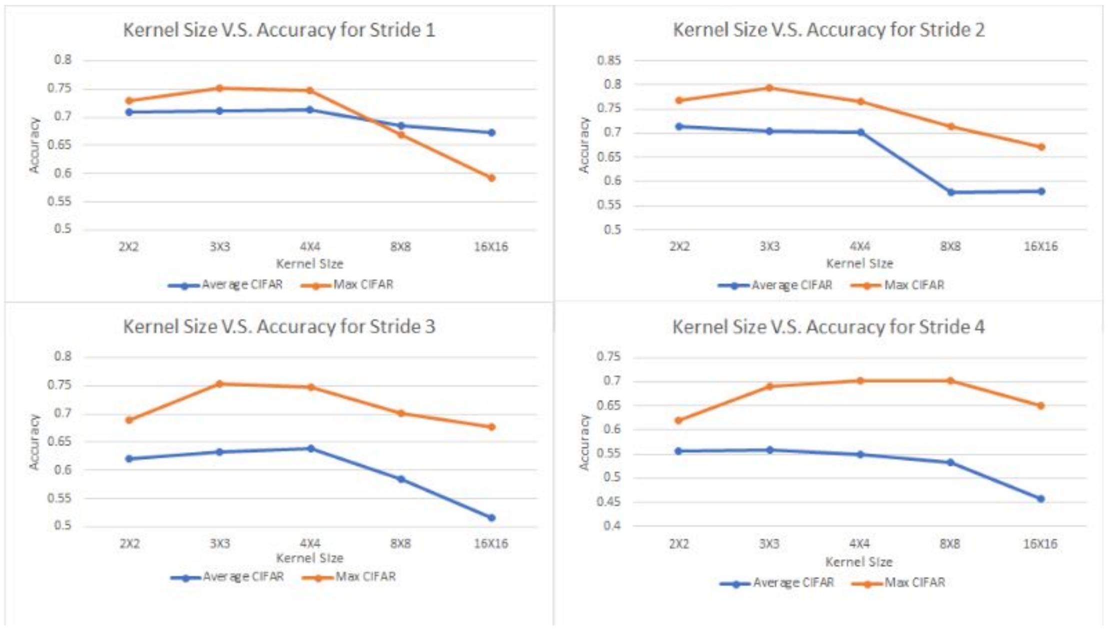


Figure 3: Effects of Pooling Size on CIFAR-10 Dataset

Figure 4, on the next page, shows the effect of pooling size on accuracy for the MNIST dataset, again not all the strides were shown, as in Figure 3. The information from the extra graphs not shown is present in Figures 1 and 2, just represented in a different form.

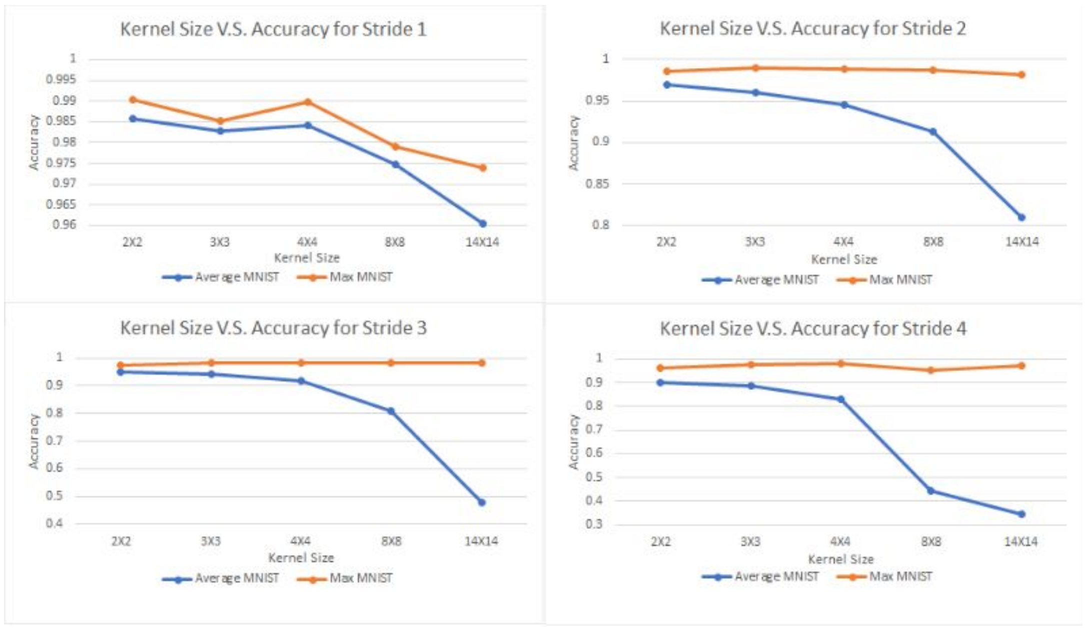


Figure 4: Effects of Stride on MNIST Dataset

Discussion:

There are a few generalized conclusions that can be drawn from the data in the results section. After discussing these observations, we will also discuss the outcome of the hypotheses outlined in the Hypotheses section earlier.

First, it can be observed that max pooling almost always outperformed average pooling. This can be observed in all of the figures above. We suspect that the reason for this is that average pooling is a linear operation, while max pooling is non-linear. In other words, average pooling is really not any different than a dense layer with some restrictive priors imposed on the weights. The models using average pooling probably did not have the capacity to successfully model the mapping between picture and label. On the other hand, max pooling is a non-linear operation, and additional non-linearities increase a model's capacity. Thus, the max pooling always had an edge over average pooling. All hope is not lost for average pooling, though. It's possible that it could work in a more complicated model where the capacity is already higher, or mixed in with max pooling layers.

Next, we will discuss the effects of stride and pool size, the other main subject of our research. Making stride too large (>=4) always resulted in harm to the test accuracy. Figures 1 and 2 both show this trend. However, there is a region of slightly improved test accuracy at size 2-3, usually for max pooling. This convex region, while only a slight improvement, indicates an area where the test accuracy improves through slight regularization. This idea of pooling as regularization will be explored more shortly. Average pooling often simply fell as stride increased, especially as pool size increased, as shown in Figure 1. As pool size increased, the range of strides that gave good results also increased; this is also expected to relate to regularization. In general, strides larger than the pool size, and really larger than 4 in general seemed to harm test accuracy. It's suspected that having a larger stride simply skips too much data, especially since the images are quite small.

As with stride, moderate pool sizes produced the best results. Figures 3 and 4 show the effects of pool size on test accuracy. Again, there appears to be a region of "correct" max pool size towards the middle, especially when examining the CIFAR-10 dataset results in Figure 3. Interestingly, the MNIST model actually seems to be robust to increases in pooling size for max pooling, once the stride is above one. With how easy MNIST is, it's possible that increasing the stride provides enough regularization that test accuracy isn't really changed by pooling size. Average pooling sometimes followed the curve evident in max pooling, but it often times fell sharply, especially on MNIST. One explanation would be that MNIST's data was too sparse, so average pooling performs poorly because it's diluted by large amounts of black pixels that give no useful information.

In general, we can see that max pooling outperforms average pooling, probably due to its introduction of non-linearities. Pool size and stride should be kept fairly small, though perhaps not equal to one. Pool sizes that were about 10-15% of the size of the image seemed to perform the best. This is just a heuristic observed on a small data set, and probably shouldn't be taken as an actual guideline without extensive testing on datasets with different image sizes. Larger pooling and stride sizes are usually not beneficial, as the higher strides lose too much data, and large pooling sizes probably dilute each observation with too much data.

Next, the hypotheses will be discussed in light of the above observations about the data. The first hypothesis was that max pooling would perform better in "sparse" data, where sparse data simply meant pictures where relatively little information actually contributed to classification. MNIST is sparse data, because it's mostly a black image, while CIFAR-10 is very colorful and information dense. The opposite was also expected, i.e. average would perform better on dense data. The intuition behind this was simply that average pooling would preserve more information than max, and that average would preserve too much on something like MNIST. Max pooling did perform better on MNIST than CIFAR, but that is certainly due to the relative ease of classifying MNIST, as average pooling also performed better on MNIST. The hypothesis that average pooling would be better on CIFAR-10 didn't hold well. There were a few times that average pooling outperformed max pooling on CIFAR-10, and max pooling always did better on MNIST. Again, average pooling doesn't seem to confer the model capacity needed to match up against max pooling. It's possible that average pooling would be helpful if *included* in a model classifying something with dense data, alongside max pooling, but it doesn't work well on its own. When average pooling did outperform max, it was generally with large pool sizes and small strides, which makes sense because average pooling accuracy should increase with the pooling size [5].

Next, it was hypothesized that pooling could act as a form of noise reduction. CIFAR-10 could be viewed as noisy, there's a large amount of small colors and shapes in a small image, and a lot of it is not precisely relevant to classifying the image. Unfortunately, there wasn't really a conclusion that could be drawn for this hypothesis. The model performed quite poorly on CIFAR-10, so there's no way to know the answer from this experiment. Potential future work could be training the same CNN on the same dataset, but inject artificial noise into one, and observing if various pooling structures conferred robustness to the injected noise.

Finally, it was hypothesized that pooling could act as regularization. Lower pooling size would reduce the amount that each data point would be used, while higher stride does the same. Stride higher than the pooling size would even skip data points entirely. This hypothesis does have some support from the data. As can be seen in the figures section (and was discussed above), very small stride and large pooling size were quite harmful to accuracy. There is a small, but noticeable and consistent, convex upside-down U shape to the test accuracy as stride or pooling size increased, especially for max pooling. There was a point where the effect was too strong (such as stride greater than the pooling size), but there was certainly some benefit for higher strides in max pooling, and smaller pool sizes. This at least lends credence to the idea that pooling size and stride could be treated as a regularization term to be tweaked.

There were some limitations to our research. First, the model underfit CIFAR-10, while also getting very strong results on MNIST. It made the results difficult to compare across the datasets. These images are also tiny-image datasets, specifically to avoid the problem of the models taking too much time to train and test. These results may not scale to larger images, and additional experimenting would be required. Also, because these were small images, the range of strides and pool sizes were limited by image size, as well as time. Future work could include simply trying a larger range of strides and pool sizes on different data sets, as well as improving the model and finding datasets that are more equal in difficulty of classification.

Conclusions:

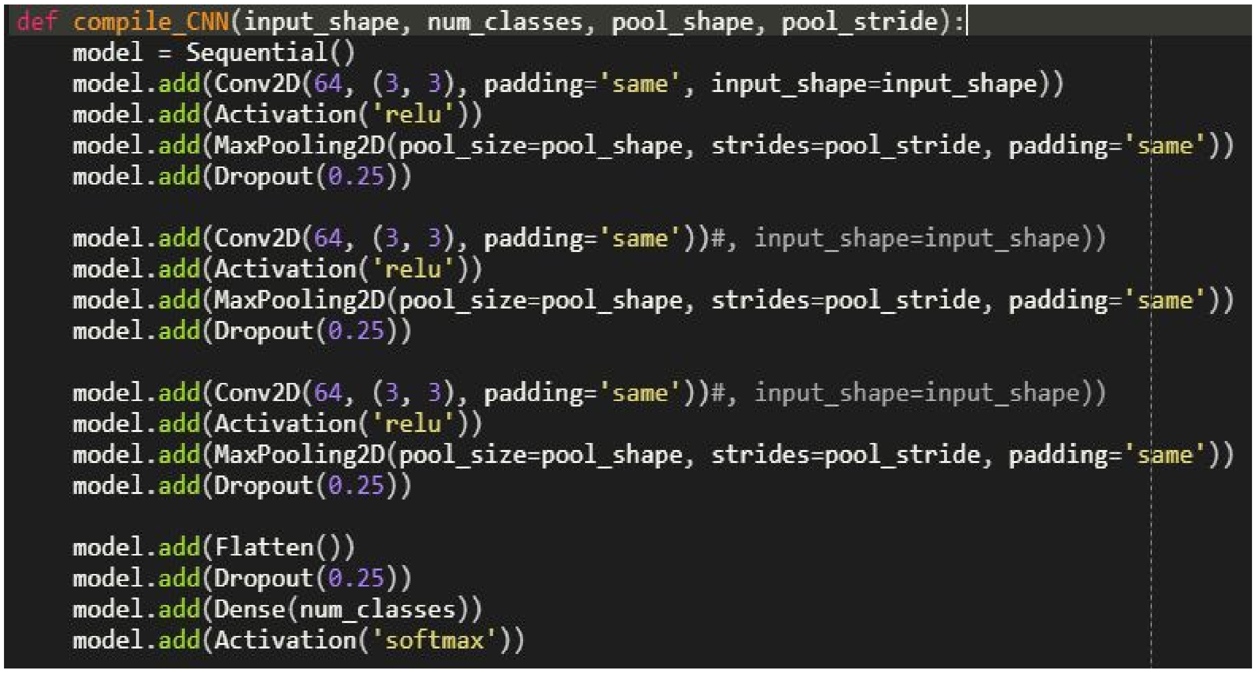
In conclusion, it was found that max pooling almost always outperforms average pooling. The main reason is believed to be that max pooling is non-linear, while average pooling is linear, and thus the model simply did not have the capacity necessary to be successful for average pooling. Average pooling did occasionally out-perform max pooling on dense data with large pool sizes, but max pooling should generally be used unless there is a specific reason that those conditions should be the case. Pooling as a filter for noise was inconclusive, and could be explored more specifically. Finally, stride and pooling size can probably be considered to be a form of regularization, where increasing stride or decreasing pooling size could improve test accuracy at the cost of training accuracy. It was observed that the best accuracy was obtained with fairly small pool sizes (2-4) and strides (2-3), which were about 10-15% of the image size. These observations are meant to be a general guide for the types and shapes of pools that should be used in a CNN.

References:

1. https://keras.io/
2. https://www.cs.toronto.edu/~kriz/cifar.html
3. https://www.kaggle.com/amarjeet007/visualize-cnn-with-keras
4. https://www.researchgate.net/figure/Example-images-from-the-MNIST-dataset_fig1_306056875

5. https://www.di.ens.fr/sierra/pdfs/icml2010b.pdf
6. Make the Github public and reference it?
7. https://research.google.com/colaboratory/faq.html

Appendix: Code (not sure if we need this)



```
def compile_CNN(input_shape, num_classes, pool_shape, pool_stride):
    model = Sequential()
    model.add(Conv2D(64, (3, 3), padding='same', input_shape=input_shape))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=pool_shape, strides=pool_stride, padding='same'))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3), padding='same'))#, input_shape=input_shape))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=pool_shape, strides=pool_stride, padding='same'))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3), padding='same'))#, input_shape=input_shape))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=pool_shape, strides=pool_stride, padding='same'))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dropout(0.25))
    model.add(Dense(num_classes))
    model.add(Activation('softmax'))
```