Generated Image Classification

Bouruc Petru-Liviu

09/01/2024

Contents

1 Introduction & Project Description

This project was about applying deeplearning methods on a classification task. The dataset contains 13000 training images, 2000 validation images and 5000 test images generated by a Generative Adversarial Networks and partitioned into 100 classes. To solve this task, I tried to use Convolutionals Neural Networks, a handpicked architecture and a consecrated architecture (ResNet18).

Besides the model, I made tests with the images unchanged and with Data Augmentation. Empirically, I found that data augmentation gives slightly better results.

2 Data Preprocessing & Augmentation

In order to make the model generalize more easily, I tried to do some data preprocessing and augmentation. I used *ImageDataGenerator* from keras. It takes as input our dataset and applies, with a random chance, our defined transformations. The transformations I used:

- zoom_range: images can take a random zoom within a defined range. From the different tests, i found that setting 0.3 as it's parameter for cropping gives the best results.
- fill_mode: defines how each cropped image is filled. *nearest* gives the best results.
- horizontal_flip: I set a small chance for randomly flip inputs horizontally.

Observing the results, for the simple convolutional neural network it didn't do much improvement, but it helped the model converge faster. For the ResNet model it brought an improvement of 0.1 - 0.2 on accuracy.

3 Simple Convolutional Neural Network

In my initial approach, I opted for a straightforward convolutional neural network architecture, given their best suited for images. This architecture is formed from multiple layers, each serving a distinct purpose.

One layer in convolutional neural networks is the $Conv2D$ layer, utilizing a filter and a specified kernel size. This layer takes over the input image, element-wise multiplying the pixels with the filter and subsequently reducing the window data size based on the kernel size. The learnable parameters reside within the filter. I opted for the ReLU activation function (y = max(x, 0)), chosen for its computational efficiency and speed.

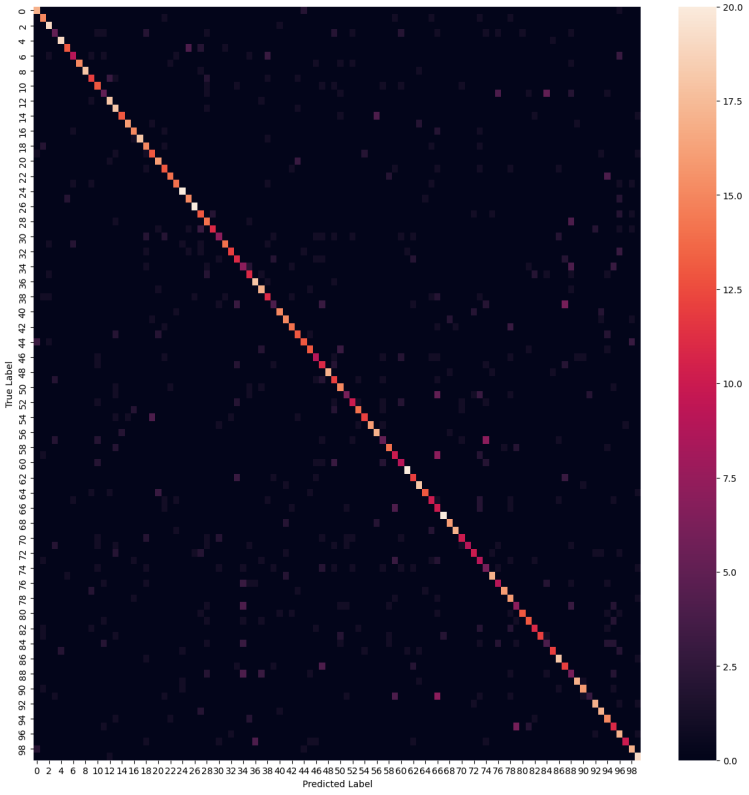
Another layer used is $MaxPooling2D$ and $AveragePooling2D$ which takes the maximum, and the average respective, value from a given window, reducing the complexity and also keeping the relevant information. Alongside this, in the architecture I also used Dropout, which have the role of randomly dropping previously learned features, reducing the risk of overfitting, and Flatten with Dense layers for liniarization, the last layer having softmax activation function which transforms the result into an array of probabilities for each class.

Additionally, I incorporated $MaxPooling2D$ layers, extracting the maximum values from specified windows. This serves to reduce complexity while retaining pertinent information. I also integrated Dropout layers, responsible for randomly excluding previously learned features, reducing the risk of overfitting. In the end, I used Flatten with Dense layers for linearization, with a softmax activation function at the last layer that transforms the output into an array of class probabilities.

I tried various combinations of these layers to identify the configuration that gives the best results. Adjusting the learning rate did not increase accuracy but had and impact on the speed of model learning: low values increased the risk of overfitting. The $Adam$ optimizer, outperforming the SGD optimizer. The highest accuracy achieved on the validation set was 0.65.

Hyperparameters tuning:

Optimizer	Learning Rate	Epoch	F1
Adam	0.01	30	0.52
Adam	0.01	40	0.52
Adam	0.01	50	0.55
Adam	0.001	30	0.59
Adam	0.001	75	0.64
Adam	0.001	100	0.65
Adam	0.0001	50	0.57
Adam	0.0001	75	0.58
Adam	0.0001	100	0.61
SGD	0.01	50	0.58
SGD	0.001	50	0.41



4 ResNet18

ResNet-18 is a specific type of deep neural network architecture which uses residual blocks (resnet blocks) in order to solve the vanishing gradient problem in very deep neural networks. It consists of 18 layers: convolutional layers, batch normalization and ReLU activation functions. There are also fully connected layers in the end.

Residual Blocks are the core idea behind ResNet is the use of residual blocks. A residual block contains a connection that allows the input to bypass one or more layers. This helps

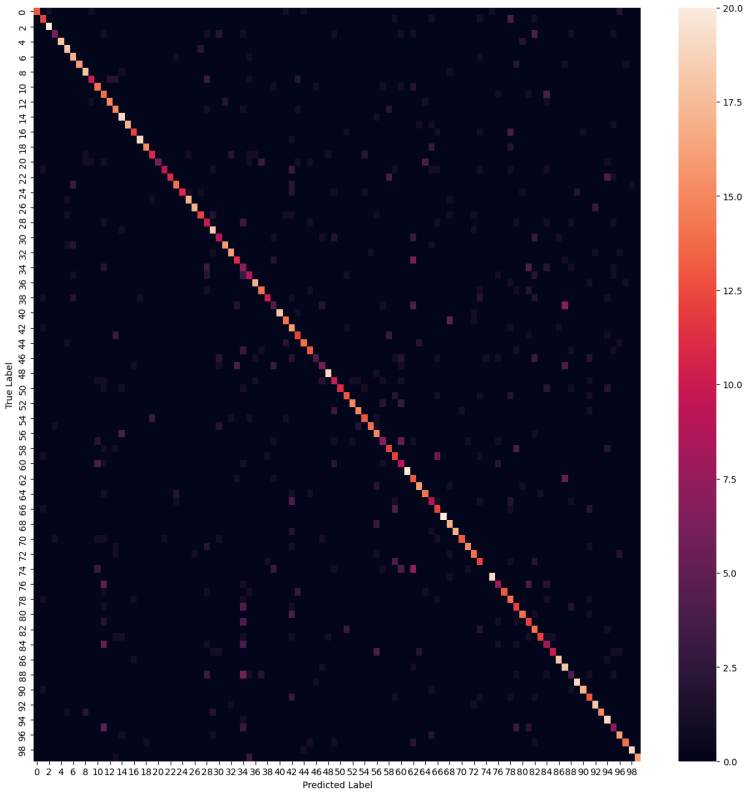
in training very deep networks by mitigating the vanishing gradient problem.

Each residual block in ResNet-18 typically consists of two convolutional layers with batch normalization and ReLU activation functions.

During the training, I tried modifying the structure by removing and adding layers. The best combination gave me an accuracy of 0.71 on private test data. I also tried adding *Dropout* layers, but they actually reduced the accuracy of the model by 0.3 - 0.4. One big improvement on the architecture was resizing the input to $(224, 224)$.

Hyperparameters tuning:

Optimizer	Learning Rate	Epochs	F1
Adam	0.01	30	0.58
Adam	0.01	40	0.62
Adam	0.01	50	0.62
Adam	0.001	30	0.70
Adam	0.001	40	0.70
Adam	0.001	50	0.71
Adam	0.0001	30	0.54
Adam	0.0001	40	0.53
Adam	0.0001	50	0.55
SGD	0.01	30	0.59
SGD	0.001	30	0.50



5 Conclusions

Taking into consideration all the tests I made, we can see that the best results were obtained using the well-known architecture ResNet, with little data augmentation, and a standard optimizer & learning rate.