Using Synthetic Image Generation for Maximizing the Performance of a Multi-label, Multi-class Image Classification Model

Reinelle Jan Bugnot and **Syed Anas Majid** and **Xiang Xinye**
School of Computer Science and Engineering
Nanyang Technological University

April 29, 2024

Abstract

In this project, we performed multi-label, multi-class, classification task on the DeepFashion Dataset by fine-tuning a pre-trained ResNeXt model. We used Optuna, a hyper-parameter tuning framework to determine the best combination of different hyper-parameters. Furthermore, apart from the traditional methods of data augmentation, we came up with a creative way to synthesize new images: using a diffusion model trained on the given dataset to generate more images of classes with fewer samples to balance the different classes. With respect to the results of the experiments we did, this method turned out to be effective.

Introduction

In this project, we used ResNeXt based model to perform a multi-label, multi-class, classification task. For our dataset, we chose the DeepFashion Dataset(Liu et al., 2016a). Based on the pre-trained model, we played multiple strategies to tune the model to improve the test accuracy on the dataset. We used an optimization framework called Optuna to test different combinations of hyper-parameters to locate the most suitable parameters for the model as well as the dataset. Furthermore, we explored several data augmentation techniques to improve the generalization ability of the model, including using the standard augmentation, which is implemented as transforms in torchvision package, as well as sythesizing our own images using a diffusion model based on the given dataset to expand our training input and mitigate the impact of severe class imbalance observed in the data.

We performed multiple experiments to verify the effectiveness of each method we took and meticulously recorded the results.

Data: DeepFashion Dataset

The DeepFashion Dataset, curated by the Chinese University of Hong Kong, stands as a pivotal resource for advancing fashion-related computer vision research. Comprising over 800,000 diverse fashion images, this extensive dataset spans a wide spectrum—from well-posed shop images to real-world consumer snapshots. What sets DeepFashion apart is its rich annotation, providing a wealth of information for each image (Liu et al., 2016b).

For our project, we specifically utilized the *Category and Attribute prediction* fashion dataset. This dataset contained 20,000 images in total, including 14,000 images as the train set, 4,000 images as test set and 2,000 images as the validation set. For each image, 6 labels are given from 6 different categories. The 6 categories are pattern, sleeve length, dress length, neckline shape, texture and fit characteristics. Hence, the machine learning task is a multi-label, multi-class, image classification task. For each class, the data distribution is showed in Fig. 3.

Model: SE-ResNeXt

The model that we used is called SE-ResNext50_32x4d, an image classification architecture featuring Squeeze-and-Excitation channel attention, ReLU activations, a single-layer 7x7 convolution with pooling, 1x1 convolution shortcut downsample, and grouped 3x3 bottleneck convolutions (Xie et al., 2017). Trained on ImageNet-1k in Apache Gluon, this architecture comprises 50 layers and a cardinality of 32 (indicating the number of groups for the grouped convolutions), with each group containing 4 channels. Fig. 1 illustrates a simple block of this network.

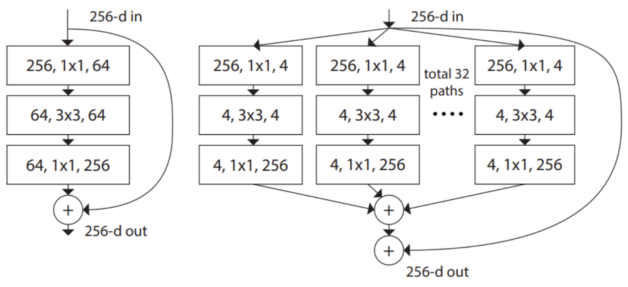


Figure 1: Left: A block of ResNet (for comparison). Right: A block of ResNeXt with Cardinality of 32 and 4 channels for each group (Xie et al., 2017)

The SE-ResNeXt50_32x4d model incorporates ”squeeze-and-excitation” (SE) blocks for dynamic channel-wise feature re-calibration. These SE blocks enhance the model's ability to focus on informative features while suppressing less relevant ones. By adaptively re-calibrating channel responses, the SE-ResNeXt50_32x4d achieves improved performance across various image classification tasks (Hu et al., 2018). In our implementation, we froze all layers of the model except for the final bottleneck block. Subsequently, we added a 512x128x26 fully connected network as the prediction head. The 26 output nodes correspond to the 26 attribute labels, which are later divided and matched into their respective 6 categories within the pipeline. This configuration results in a total of 17,242,394 trainable parameters.

Experiments

Loss Function

The Loss Function used for the project is the *weighted cross-entropy loss*. The 26 output logits of the model is sliced into 6 categories whose lengths are [7, 3, 3, 4, 6, 3], corresponding to the number of classes per label category in the data. The weights used for the cross-entropy loss are tensors of the same length, whose values are calculated as follows:

$$w_i = \frac{250}{11N_i} \tag{1}$$

where w_i is the weight of class i for all i from 1 to 26, and N_i is the total population of class i in the training set. This expression is derived from the inverse of the class distribution, which effectively gives more weight to low frequency classes and vice versa, helping to mitigate the effects of the class imbalance observed in the dataset.

Tunable Parameters

Across our experiments, we used 5 different regularization techniques, which includes: (1) weight decay, (2) label smoothing, (3) dropout, (4) cosine annealing, and (5) different data augmentation techniques, as well as 4 optimization strategies including (1) tuning the Learning Rate, (2) Learning Rate warmup, (3) tuning the Batch size, and (4) tuning the Adam Optimizer Beta parameters. From here, we maintained the following experiment settings constant across all of our trials:

- 60 Epochs for Training, 25 epochs per trial for Tuning using a hyperparameter optimization framework.
- Cosine Annealing to dynamically adjust the learning rate during training.
- Learning Rate warm-up set to 5 to gradually increase the learning rate from 0 to it's starting value within the first 5 epochs, before decaying via Cosine Annealing.
- Adam as optimizer

These baseline settings ensures that any observed differences in performance were primarily due to our parameter tuning and data augmentation experiments. This leaves us seven tunable hyperparameters:

1. **Learning Rate** (α): The step size for adjusting model weights during training.
2. **Weight Decay**: A regularization technique that penalizes large weight values to prevent overfitting.
3. **Label Smoothing**: A method that introduces a small amount of uncertainty in the ground truth labels.
4. **Dropout**: A regularization technique that randomly drops out a fraction of neurons during training.
5. β_1 **and** β_2 **(Adam)**: These control the exponential moving averages of gradients and squared gradients, of the Adam Optimizer.
6. **Batch Size**: Optimal selection means a balance between computational efficiency and model performance.

Table 1 summarizes the hyperparameter settings and their corresponding parameter search ranges.

Table 1: Hyperparameter Settings and Ranges

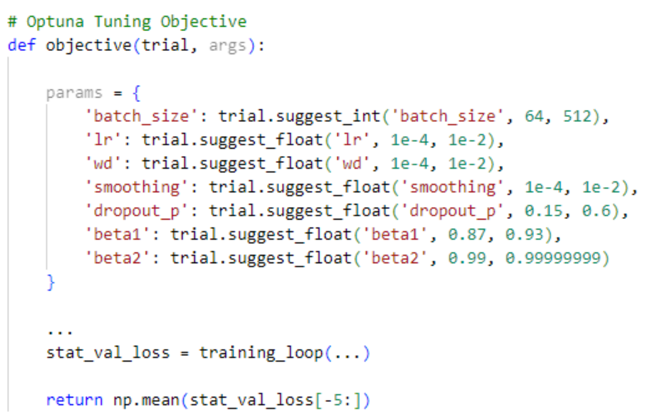
Hyperparameter	**Range**
Learning Rate (α)	[0.0001, 0.01]
Weight Decay	[0.0001, 0.01]
Label Smoothing	[0.0001, 0.01]
Dropout Rate	[0.15, 0.6]
β_1 (Adam)	[0.87, 0.93]
β_2 (Adam)	[0.99, 0.99999999]

Hyperparameter Tuning using Optuna

Hyperparameter tuning is the process of finding the optimal set of hyperparameters for a machine learning model, with the goal of improving a model's predictive accuracy and generalization ability. We can perform this manually by iteratively adjusting the hyperparameters based on the model's performance on a validation set. This approach, however, can be time-consuming and may not lead to the best results due to the high dimensionality of the hyperparameter space.

In this project, we implemented a hyperparameter optimization framework called **Optuna**, which is an agnostic framework that optimize the values based on searching and pruning strategies. At its core, it operates using Bayesian optimization with an algorithm called TPE (Tree-structured Parzen Estimator) (Akiba et al., 2019). Optuna facilitates the automatic hyperparameter tuning for our models during the experiments. This framework allows a more efficient search for hyperparameter values by monitoring the intermediate evaluation result of the model and pruning the unpromising trials to speed up the exploration. Optuna will conduct multiple explorations based on the user defined range of potential hyperparameter values and evaluate the results against the defined evaluation metrics (Akiba et al., 2019). Shown in the Fig. 2 is an example set-up of the Optuna objective() function.

The Optuna objective function contains the *trial* method that systematically selects parameter values based on a given range specified using the `.suggest_float()`, `.suggest_int()`, and `.suggest_categorical()` functions. A narrow suggestion range will yield relatively more consistent performance, but limits the optimization capacity of the model. Hence, identifying the proper trial range



```
# Optuna Tuning Objective
def objective(trial, args):

    params = {
        'batch_size': trial.suggest_int('batch_size', 64, 512),
        'lr': trial.suggest_float('lr', 1e-4, 1e-2),
        'wd': trial.suggest_float('wd', 1e-4, 1e-2),
        'smoothing': trial.suggest_float('smoothing', 1e-4, 1e-2),
        'dropout_p': trial.suggest_float('dropout_p', 0.15, 0.6),
        'beta1': trial.suggest_float('beta1', 0.87, 0.93),
        'beta2': trial.suggest_float('beta2', 0.99, 0.99999999)
    }

    ...
    stat_val_loss = training_loop(...)

    return np.mean(stat_val_loss[-5:])
```

Figure 2: Optuna Objective Function (simplified)

values is crucial for the successful implementation of Optuna.

Data Augmentation

In this project, we applied data augmentation in both the standard fashion and the novel fashion. In the beginning, we tested our model with transform-based methods, as listed in the 2, only to find the performance to be unsatisfactory. Then, in order to expand the dataset and introduce more variety and correct for class imbalances, we came up with the idea to synthesize new images with existing images and add into the dataset.

Standard Augmentation We adopted several standard augmentation techniques:

- **Random horizontal flip:** Perform horizontal flip on images randomly. We set the possibility to be 0.5.
- **Color jitter:** Introduce random color variations to the images.
- **Zoom in:** We use center-crop to perform the zooming in of images. We perform this technique because we noticed that many images had pure-color background, and cutting off the edging blank space can reduce the impact of background on the data.
- **Normalize:** We use the mean and standard deviation of Imagenet-1k dataset, which is the dataset that the pre-trained part of our model was trained on. After normalization, our dataset would show more similarity to the Imagenet-1k dataset, which is more familiar to the pre-trained model we are using.
- **Random affine:** Perform random affine transformations to input images. An affine transformation is a linear mapping that preserves points, straight lines, and planes.

These operations were directly performed on the input images. They were applied to the synthesized images as well when we added them to the dataset.

Synthetic Image Generation

Manually Implementing Augmentations The most natural thought would be to change some part of the original images to make it look slightly different to improve the generalization ability of the model. So we increased the contrast and applied the sharpen filter to modify the color space of the image. Then we added the modified images back to the dataset. The generated images still preserve most of the features of the original image, so this method can only have limited impact on the performance. Some examples can be seen in Fig. 4 (Left).

Synthesize New Images With a Model During the data exploration stage, we observed that there was a severe class imbalance in the dataset. The figure below shows the class counts against the class label attribute.

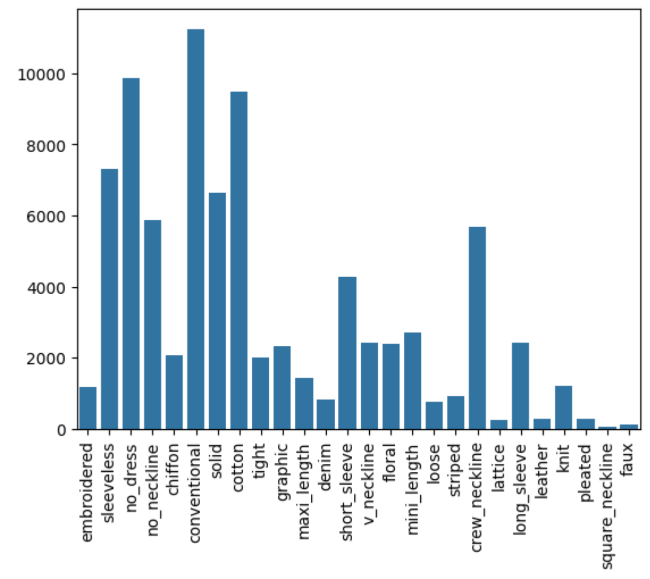


Figure 3: Count Distribution of Attributes in the DeepFashion Dataset.

We noticed that the following class attributes had under 500 samples: 'lattice', 'leather', 'pleated', 'square_neckline' and 'faux'. To tackle this issue, we explored the use of a Diffusion model to generate synthetic data for the classes, since it has been shown to improve performance on classification tasks (Azizi et al., 2023).

Diffusion models are an equally-weighted sequence of denoising autoencoders, which function by introducing Gaussian Noise successively, and then learning to recover the original image from the noisy data. The model's sequential nature allows the learned process to be reversed into the forward process, in which a sample of noise can be refined into an image. Samples at each phase are drawn from a learned Gaussian distribution $\mathcal{N}(x_t; \mu(x_t, t), \sum_\theta(x_t, t))$ where the mean of the distribution $\mu(x_t, t)$ is centered on the distribution in directly preceding phase. The variance of the distribution follows a fixed schedule. For us to be able to focus on the classes that we need to generate, we employed a conditional diffusion model, which employs text-guided control features by matching text to a sequence of CLIP embeddings, and mapping these to images in the training process (Radford et al., 2021).

Table 2: Data Augmentation Configuration

Aug Type	Augmentation Steps
basic	hflip, norm
soft	hflip, norm, zoom, colorjitter, random affine
hard	hflip, norm, zoom, colorjitter, random affine, sharpness, contrast
synthetic	hflip, norm, zoom, colorjitter, random affine, sharpness, contrast, add custom generated images

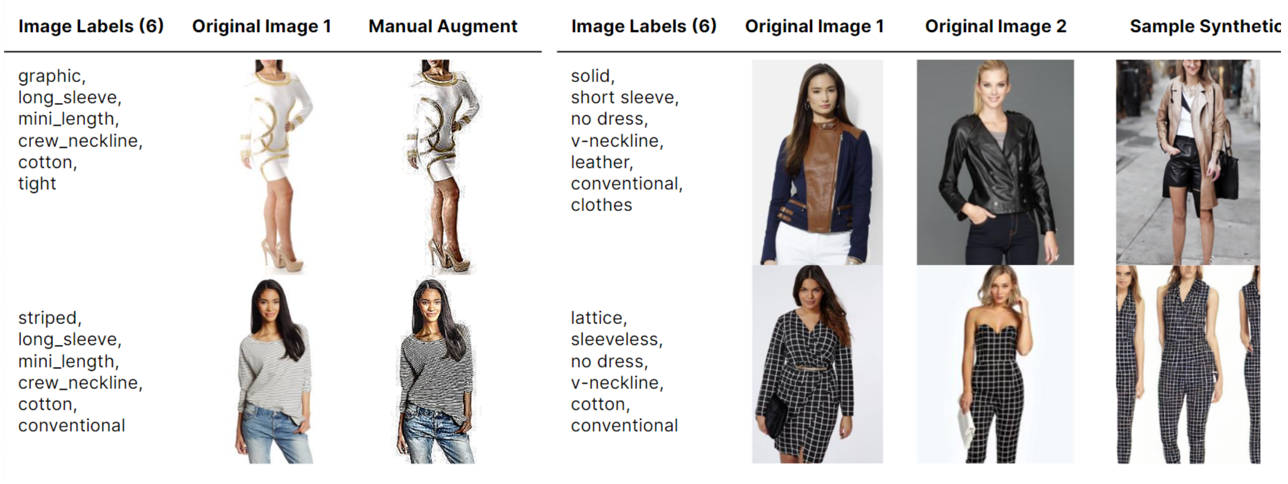


Image Labels (6)
Original Image 1
Manual Augment
Image Labels (6)
Original Image 1
Original Image 2
Sample Synthetic
graphic, long_sleeve, mini_length, crew_neckline, cotton, tight
solid, short sleeve, no dress, v-neckline, leather, conventional, clothes
striped, long_sleeve, mini_length, crew_neckline, cotton, conventional
lattice, sleeveless, no dress, v-neckline, cotton, conventional

Figure 4: Manually augmented image samples for the *hard* augmentation type in Table 2 (left). Sample generated synthetic images (2 out of 1,236) using Stable Diffusion. These images are added to the training data as part of our data augmentation strategy (right).

We generated training data by extracting images that belong to the classes stated. The images were then tagged to a caption that was created based on the labels of the patricular image. An example is shown below:

```
{"file_name": "Grid_Print_Jumpsuit_img_
00000048.jpg",
"caption": "A person wearing lattice
sleeveless no dress v neckline cotton
conventional clothes"}
```

The ”file_name” is the name of the image file and the ”caption” field was programmatically generated. The captions are the attributes of the image in the dataset, with the added prefix ”A person ” and the suffix ” clothes”.

We chose a pre-trained Stable Diffusion model for this task. The model weights were initialized to the first published checkpoint from HuggingFace (CompVis, 2022) - ”stable-diffusion-v1-1”. The model was then fine-tuned on our data-subset and generated captions for 1500 steps on a machine with a single RTX4090 GPU. Inference of the fine-tuned model was conducted by regenerating captions for the imbalanced classes using the same method and performing a forward pass. The mapping of the captions was kept track of to convert it back to the same format as that of the dataset. Fig. 4 shows a few examples of the generated synthetic images from their given six class labels. We can see from the figure that the diffusion model was able to generate clothing images that accurately represent the class labels, as seen by the style similarity compared to the provided reference images.

Results

Table 3 shows the consolidated results across all of our experiments. We first focused on tuning the base model hyperparameters without heavy data augmentations (experiment indices 1 to 10). Here, we implemented two strategies. The first strategy is by methodically tuning the parameters, one at a time. We did this by running 5 trials with 25 epochs each, using Optuna, and then selected the parameter value that generated the best results calculated by the mean of the validation losses of the final 5 epochs. For example, in experiment 1, we focused on optimizing the learning rate without any regularization. In experiment 2, we used the best learning rate from the previous experiment, and then optimized on weight decay parameter. In experiment 3, we kept the best learning rate and weight decay parameters, and optimized on label smoothing, and so forth, until experiment 7.

Figure 5 illustrates the impact of each individual tuning experiment on the training and validation curves for experiment indices 1 to 7. We observed that simple regularization strategies helped mitigate overfitting. Without regularization, a significant gap existed between the training accuracy and validation accuracy (and/or loss), with the training accuracy converging at around 98% while the validation accuracy remained at around 60%. However, as we incorpo-

Table 3: Experimental Results

index	batch_size	lr	wd	smoothing	augmentation	dropout	β_1	β_2	Test Loss	Test Acc
1	256	9.453E-3	0.000	0.000	basic	0.000	0.900	9.990E-1	3.46	65.40%
2	256	9.453E-3	8.875E-4	0.0000	basic	0.000	0.900	9.990E-1	1.111	65.44%
3	256	9.453E-3	8.875E-4	9.311E-3	basic	0.000	0.900	9.990E-1	0.825	65.73%
4	256	9.453E-3	8.875E-4	0.0000	soft	0.000	0.900	9.990E-1	0.785	67.42%
5	256	9.453E-3	8.875E-4	9.311E-3	soft	0.000	0.900	9.990E-1	0.743	66.64%
6	256	9.453E-3	8.875E-4	9.311E-3	soft	0.461	0.900	9.990E-1	**0.723**	**68.07%**
7	359	9.453E-3	8.875E-4	9.311E-3	soft	0.461	0.900	9.990E-1	0.728	67.84%
8	140	3.295E-3	8.100E-4	2.500E-3	soft	0.450	0.891	9.923E-1	**0.741**	**68.58%**
9	256	3.032E-3	2.730E-3	4.369E-4	soft	0.266	0.900	9.990E-1	0.795	66.81%
10	256	3.032E-3	2.730E-3	4.370E-4	soft	0.266	0.900	9.990E-1	0.779	66.72%
11	256	3.032E-3	2.730E-3	4.370E-4	hard	0.266	0.900	9.990E-1	0.765	67.28%
12	256	9.453E-3	8.875E-4	9.311E-3	synthetic	0.461	0.900	9.990E-1	0.714	66.98%
13	324	7.166E-3	3.935E-3	1.677E-3	synthetic	0.174	0.910	9.911E-1	0.696	69.74%
14	76	6.256E-4	7.580E-4	5.438E-3	synthetic	0.165	0.909	9.901E-1	**0.674**	**70.89%**

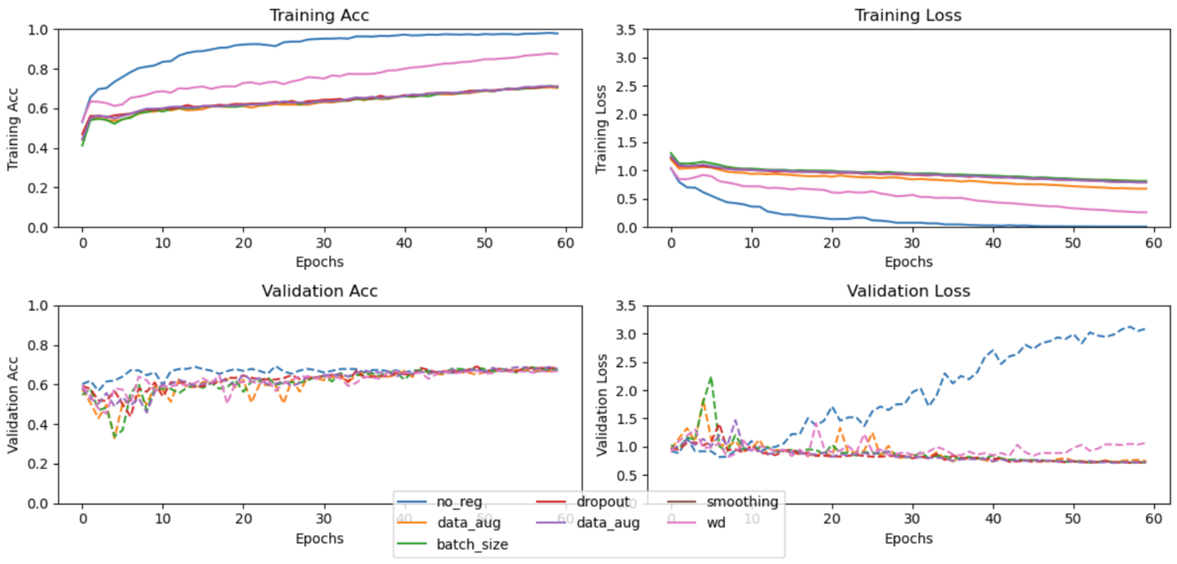


Figure 5: Training and Validation Curves for some notable experiments during individual tuning.

rated and properly tuned regularizers (and some occassional optimization techniques) into the training setup, we achieved progressively better results, as evidenced by the steadily improving test scores. Notably, from this strategy alone, we achieved a test accuracy of **68.07%** as observed in experiment 6 when we added dropout.

The second strategy is a brute force search that fully utilizes the Bayesian parameter optimization strategy through Optuna. Instead of tuning the parameters individually, we defined a parameter search range for each hyperparameter (Table 1) and ran Optuna for 15 trials at 25 epochs each. Afterwards, we recorded the best parameters and re-trained the model for 60 epochs using those parameters, the results of which are shown in experiments 8 to 14 in Table 3. Each experiment ran for approximately 5-8 hours. From the test results we can see that the best performing model, prior to adding more potent data augmentations, was achieved in experiment 8 with a test accuracy of **68.58%**.

Experiments 11 to 14, as detailed in Table 3, explores the impact of augmented data strategies on model performance. Notably, we augmented the training data with synthetic images generated from the image labels themselves. Experiment 11 specifically explored the effects of additional data augmentation steps beyond those already implemented in experiment 10 (and prior). The outcome was a promising test accuracy of 67.28%, surpassing the reference experiment's yield of 66.72

Subsequently, we incorporated synthetic images generated using Stable Diffusion into the training dataset. Experiment 12, utilizing the exact parameters from experiment 6, exhibited a performance decline—from 68.07% to 66.98%. This result aligns with expectations, given that the parameters in experiment 6 were meticulously tuned for the original dataset. Hence, for experiments 13 and 14, we performed a brute force search strategy on all tunable parameters using the synthetic data on top of all previous data augmentations. Remarkably, these experiments achieved test accuracy values of 69.74% and **70.89%**, respectively—both surpassing all prior experiments by a substantial margin (with a difference of over 1% in test accuracy)."

Conclusion

This project demonstrates the critical role of hyperparameter tuning and regularization in enhancing model performance. Our approach, which combined systematic parameter adjustments and careful regularization, produced substantial improvements to our model's performance. Initially, without proper regularization, the training accuracy was very high while the validation accuracy lagged significantly behind; indicating an undesirable case of overfitting. As we introduced regularization techniques and properly optimized the hyperparameters, the gap narrowed, mitigating overfitting and leading to better generalization. Our workflow was significantly streamlined by integrating Bayesian parameter search using Optuna, which further expanded our exploration of the parameter space instead of manually tuning the values ourselves. However, simply tuning the base model hyperparameters has its limitations. The real breakthrough in maximizing our model's performance came with the incorporation of synthetic images generated from Stable Diffusion. This addition significantly enhanced test results, surpassing all previous experiments. These findings underscore the importance of both meticulous parameter tuning and proper data augmentation in achieving superior model performance.

In conclusion, our project highlights the synergy between hyperparameter optimization and regularization. By carefully tuning the model parameters and integrating appropriate data augmentations, we are able to achieve remarkable results. Future work could explore additional data augmentation techniques and investigate the impact of different optimization algorithms.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. *arXiv preprint arXiv:1907.10902*.

Azizi, S., Kornblith, S., Saharia, C., Norouzi, M., and Fleet, D. J. (2023). Synthetic data from diffusion models improves imagenet classification.

CompVis (2022). Stable diffusion. `https://huggingface.co/CompVis/stable-diffusion`.

Hu, J., Shen, L., Albanie, S., Sun, G., and Wu, E. (2018). Squeeze-and-excitation networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 42(8):2013–2023.

Liu, Z., Luo, P., Qiu, S., Wang, X., and Tang, X. (2016a). Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Liu, Z., Luo, P., Qiu, S., Wang, X., and Tang, X. (2016b). Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. (2021). Learning transferable visual models from natural language supervision.

Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. (2017). Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*.