Deep Learning-Based Approaches for Automatic Mango Grading

B06401036 Hsiao-Yen Tung [*1] **B06401078 Yu-Lun Hsu** [*1] **B06902080 Shih-Lun Wu** [*1]
Team Name: html_MadMangoMorons

1. Introduction

This work is coupled with Taiwan AI CUP 2020[1], which aims to help mango growers effortlessly classify their produce into grade A, B, or C with AI algorithms. The dataset provided contains 7200 mango images of varying quality, out of which 5600, 800, 800 are in training, validation, and public test set respectively. In Section 2 we set up a simple baseline; in Section 3 we elaborate our data preparation process; Section 4 presents the end-to-end CNNs we use; we introduce a novel convolutional autoencoder-classifier architecture in Section 5; and, in Section 6, the performance of our models and some observations are discussed.

The packages used for the project include: `torch`[2] (PyTorch), `torchvision`[3], and `scikit-learn`[4]. Since transfer learning has been proved effective in carrying learned knowledge from general domains to specific ones having limited data, we also take advantage of the models pretrained on ImageNet, a 1000-class, large-scale image corpus. The weights and guidelines for using the pretrained are provided by the `torchvision` package.

2. Baseline Effort

Our baseline model consists of 3 separate components working in tandem: a pretrained AlexNet (Krizhevsky et al., NIPS 2012) for feature extraction; a vanilla autoencoder for dimensionality reduction; and, an XGBoost (Chen & Guestrin, ACM SIGKDD 2016) gradient boosting decision trees classifier that handles the prediction.

To begin with, we feed the images "as is" (i.e., without any preprocessing other than resizing to $(224, 224)$ and channel normalization according to `torchvision` guidelines) into the AlexNet to obtain a 9216-dim feature vector. The resulting feature vectors are then used to train a single-hidden-layer vanilla autoencoder that reduces the feature dimensionality to 512. Finally, we train an XGBoost classifier taking the compressed features as input and minimizing the multi-class cross-entropy loss. The training details can be found in Table 1. Performance-wise, this model scores an accuracy of 74.5% on the validation set, and 73.3% on the public test set.

Part	Attribute	Detail
Autoencoder	*activation*	LeakyReLU, a=0.04
	optimizer	Adam, lr=5e–3
	# epochs	Early-stopping on val. loss, w/ patience=20
XGBoost Clf.	*tree depth*	4
	feature subsampling	25% per tree
	lr	0.01
	# epochs	400, early-stopping w/ patience=50

Table 1. Details of the baseline model. The hyperparameters of XGBoost (i.e., tree depth & feature subsampling rate) are chosen via grid search under 5-fold cross-validation.

3. Data Preparation

The quality of data is often the most crucial aspect of machine learning tasks, hence we try out various ways to enhance it. These efforts include selecting the most suitable input image size, shifting or normalizing the pixel values, removing the irrelevant image background with image segmentation algorithms; and, increasing the effective dataset size with various data augmentation techniques.

3.1. Image Size

The input to our models is a $(224, 224)$ RGB image. We first try to input smaller images such as $(128, 128)$, or even $(64, 64)$ ones. However, the result does not turn out well. Due to interpolation, some tiny defects critical for grading would disappear in the resizing process. Therefore, we

[*]Equal contribution [1]Department of Computer Science and Information Engineering, National Taiwan Univ., Taipei, Taiwan.

Final Project Report, *Machine Learning Techniques*, Spring 2020 (Prof. Hsuan-Tien Lin), National Taiwan Univ., Taipei, Taiwan, 2020.

[1] `aidea-web.tw/topic/72f6ea6a-9300-445a-bedc-9e9f27d91b1c`
[2] `pytorch.org/docs/stable/index.html`
[3] `pytorch.org/docs/stable/torchvision/index.html`
[4] `scikit-learn.org/stable/user_guide.html`

decide to stick with the input size $(224, 224)$.

3.2. Pixel Value

By default, the value of each pixel of a channel lies from 0 to 1 after converting the image to a `torch` tensor. Subsequently, we consider 3 different techniques when dealing with the pixel values, listed as follows:

1. **subtract 0.5 from each pixel**: This step keeps the value of every pixel range between –0.5 and 0.5, which makes the training more stable.
2. **normalize w.r.t. our dataset**: We compute the RGB mean and standard deviation on the training dataset, and use them to normalize the images.
3. **normalize for pretrained models**: According to `torchvision` guidelines, before training with the pretrained models, the images should be normalized with mean=[0,485, 0,456, 0,406] and standard deviation=[0,229, 0,224, 0,225].

For non-pretrained models, the input values are processed by the first method. For pretrained models, the input values are processed by the third method. We don't recommend the second one, since it leads to the worst performance.

3.3. Image Segmentation

Concerning background removal, we try 2 different segmentation methods, one is the non machine learning-based Canny edge detection algorithm (Canny, PAMI 1986), the other is the Mask R-CNN (He et al., CVPR 2017). Since Canny edge detection segmentation only performs well on a small portion of data containing simple background, we adopt Mask R-CNN as our final solution. Mask R-CNN is an enhanced version of the Faster R-CNN (Ren et al., NIPS 2015), both being robust methods for object detection. We modify the open-source codes[5] to add the "mango" category. The following are the steps we perform:

1. We annotate our dataset. 100 images are annotated, of which 60 are used as training data and 40 are kept as validation data. Although ImageNet does have a "mango" category, it gives unsatisfactory segmentation results on our data. Hence, we utilize the VGG Image Annotator (Dutta & Zisserman, ACM MM 2019) to mark the mangoes' positions with polygons for further fine-tuning.
2. We fine-tune Mask R-CNN on the 60-image training set. We initialize the model with ImageNet pretrained weights. We assume the first few layers of the network are already well-trained to extract low-level features, hence we freeze their weights and only allow the last layers to be updated. We get the best result with 20 fine-tuning epochs and learning rate set to 1e–3.
3. We perform image segmentation. At first, we use a splash method to extract mangoes from images. What splash method does is finding the exact boundary of the mango. However, our classification models perform not as desired with these data, most likely due to the rugged outline of the extracted mangoes. Hence, we finally use the bounding box method. The bounding box is obtained from the extreme points of the border given by the splash method. We find that entire mangoes can be better preserved with bounding boxes.

[5] `github.com/matterport/Mask_RCNN`

Figure 1. Background removal results. The left image uses splash method, with which a rugged boundary of the mango is obtained. The right image uses bounding box method, with which the whole mango is preserved.

3.4. Data Augmentation

In each training epoch, we randomly apply the following set of perturbations on the images to augment our data:

- **Horizontal or vertical flip**, each with probability=0.5;
- **Brightness**, –20 to +20 %;
- **Contrast**, –10 to +10 %;
- **Rotation**, –20 to 20 degrees;
- **Zoom in/out**, 0.8x to 1.25x.

All of these methods are readily available in the `torchvision` package.

4. End-to-End CNNs

Owing to their tremendous success on the ImageNet corpus, we adopt the following well-known CNNs for our work: AlexNet, VGG11 (w/ batchnorm), VGG16 (w/ batchnorm), and ResNet34. Besides, we also implement a CNN based on VGG16 to see if it better suits our task. This section introduces the adopted end-to-end CNNs and lists the details for training (or fine-tuning) them. All of the end-to-end CNNs optimize the multi-class cross-entropy loss.

4.1. AlexNet

AlexNet (Krizhevsky et al., NIPS 2012) is the very first successful CNN on the ImageNet dataset. AlexNet contains 5 convolutional layers and 3 fully-connected layers. Dropout, ReLU, and max-pooling are also used in the architecture.

Training details. We initialize the model with pretrained weights, and replace the last fully-connected with a new one for our task (output dimension=3). The last layer's weights are initialized with He initialization (He et al., CVPR 2015) and the bias is filled with 0. The dropout rate for fully-connected layers is fixed at 0.5, since by our experiment, changing the rate doesn't improve the performance. The model is trained for 70 epochs. Besides, We choose 32 as the batch size, SGD w/ momentum as the optimizer, and "StepLR" as the learning rate scheduler, which decays the learning rate by 10% every 20 epochs. The learning rate is initially set to 1e–3, and the momentum is set to 0.9.

4.2. VGGs

VGG (Simonyan & Zisserman, ICLR 2015) makes improvements over AlexNet by replacing large-sized convolutional kernels with multiple 3x3 and 1x1 kernels. VGG has 5 "VGG blocks", which are composed by a sequence of convolutional layers, ReLU layers for nonlinearity, and a max-pooling layer. Besides, VGG11 and VGG16 are named according to the number of convolutional and fully-connected layers used in the model.

4.2.1. Model implementation based on VGG backbone with small variations

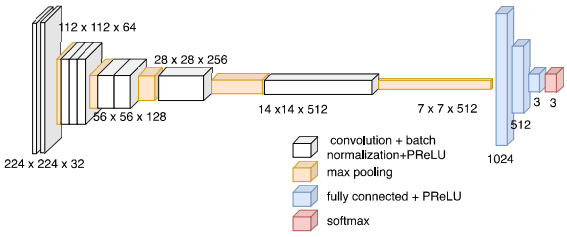


Figure 2. The architecture of our model implementation based on VGG backbone with small variations. Please note that we use parametric ReLU (PReLU) instead of ReLU as the activation function.

In our own model (refer to Fig. 2), we make some changes to the original VGG architecture. Because in our task we only need to classify the mangoes into three classes, not as complicated as the ImageNet (1000 classes), we reduce the number of convolutional layers. Also, the dimensions of the 3 fully-connected layers are changed to 1024, 512, and 3 respectively. We hope in this way, we can train the model faster even without using pretrained weights. In addition, parametric ReLU (PReLU) is used in all hidden layers.

Training details. For optimization, We use Adam optimizer in the first 100 epochs and SGD in the last 50 epochs. The learning rate is set to 1e–4, and the batch size 32. The model is regularized using dropout for the first two fully-connected layers with a 0.5 dropout rate. We initialize the weights with He initialization and fill the bias with 0.

4.2.2. VGG11 and VGG16

Training details. We try 3 ways in training VGGs. First, we train VGGs from scratch to see whether it is better to use the original VGGs than our own model (ref. Sec. 4.2.1). The other two methods are training models with pretrained weights: one is fine-tuning the entire model; the other is freezing the first few layers. We train the VGGs for 35 epochs. The batch size is set to 32. We use the SGD w/ momentum optimizer with initial learning rate set to 5e–3, and momentum set to 0.9; and, "StepLR" is adopted as the learning rate scheduler.

4.3. ResNet34

ResNet (He et al., CVPR 2016) utilizes shortcut connections to solve problems often encountered when training deeper neural networks. ResNet34 has 4 sub-modules which consist of 3, 4, 6, and 3 basic blocks respectively. A basic block is composed of 2 convolutional layers with batch normalization and ReLU activation. It is called ResNet34 for having 33 convolutional layers and 1 fully-connected layer.

Training details. Similar to the aforementioned settings, we replace the last fully-connected layer with a new one for our task. The model is trained for 30 epochs. We also use batch size 32, SGD w/ momentum optimizer, and the "StepLR" learning rate scheduler, with the initial learning rate set to 1e–3 and momentum set to 0.9.

5. Convolutional Autoencoder-Classifiers

In addition to end-to-end CNN classifiers, we approach this task with another network structure, which involves:

- A convolution-based **encoder** that compresses an image into a latent vector;
- A convolution-based **decoder** that reconstructs the image from the latent vector and some intermediate encoded features;
- A fully-connected **classifier** that takes the latent vector as input and gives the class prediction.

The rationale behind this solution is as follows: first, the presence of autoencoder forces the network to preserve the essential information for reconstruction, rather than solely cramming for the prediction task, thereby reducing the overfitting of the classifier; second, the compressed latent features also provide more flexibility for downstream tasks, for example, if we prefer doing a "type of defect" classification instead, we can keep the architecture and weights of the autoencoder intact, replace the classifier part with a task-specific one, and simply fine-tune the network for the task. In this sense, whatever the prediction task is, the network can take advantage of what it has learned from all the mango pictures it's seen, instead of being confined to a single task.

5.1. Network Architecture

Our implementation is based on the open-source codes[6] for the networks presented in a previous work on angiodysplasia (an intestinal disease) detection and localization (Shvets et al., ICMLA 2018). In that work, 3 encoder-decoder network architectures were proposed, with the main difference lying in their pretrained encoders:

- **TernausNet11**—contains VGG11 encoder;
- **TernausNet16**—contains VGG16 encoder;
- **AlbuNet34**—contains ResNet34 encoder.

We revamp the networks to suit our task and dub them **Ternaus11Clf**, **Ternaus16Clf**, and **Albu34Clf** respectively. Figure 3 is an illustration of the **Ternaus16Clf**'s architecture (the other 2 networks are similarly-structured). For each convolutional block in the encoder, there is a corresponding decoder deconvolutional block, which takes its input not only from its preceding block, but also from the skip connection linked to the convolutional block. These deconvolutional blocks are in charge of reconstructing the image, layer after layer, from the compressed latent representation and the intermediate features sieved by the convolutional blocks. Working in parallel to the decoder is the fully-connected, LeakyReLU-activated classifier of dimensions d-1024-128-3, where d is the latent feature dimension, which we append for prediction.

5.2. Training Details

Since the networks contain both an autoencoder and a classifier, a hybrid loss is required for optimization, one part of which being the reconstruction loss L_{rec}, which is the mean squared (MSE) error between the reconstructed image and the input image; the other part is the classification loss L_{clf}, which is the same cross-entropy loss applied to end-to-end CNNs. The hybrid loss is obtained via $L = \alpha L_{rec} + (1 - \alpha) L_{clf}$. Through experiments, we find that the autoencoder part is quite robust, probably due to the pretraining of the encoder and the decoder's access to intermediate features; therefore, we pick α=0.05 for a balanced optimization.

We choose 64 as the batch size, Adam as the optimizer, and "ReduceLROnPlateau" scheme as the learning rate scheduler, which sets the initial lr=2e–4, and decays the learning rate by 80% whenever the validation accuracy hasn't improved for 8 epochs. The training process is terminated by early-stopping with 20 epochs of patience on the improvement of validation accuracy. Furthermore, a 0.4 dropout rate is applied on the fully-connected classifier. The entire training process, for the 3 networks alike, takes about 2 hours on an NVIDIA V100 GPU with 32GB memory.

[6] github.com/ternaus/angiodysplasia-segmentation

6. Experiments and Discussions

In this section, we present and compare the performance of our proposed models, as well as some ensemble methods adopted to achieve higher accuracy. Last but not least, we provide some insights into the misclassified samples by VGG16, the best-performing architecture in our work.

6.1. End-to-End CNNs

Model	Accuracy	
	Train	Val.
our own CNN	81.3 %	80.1 %
non-pretrained VGG16	83.7 %	80.5 %
pretrained AlexNet	80.9 %	81.3 %
pretrained VGG11	82.2 %	83.1 %
pretrained VGG16	83.8 %	**83.6** %
pretrained ResNet34	86.7 %	82.4 %
pretrained AlexNet w/ im. seg.	81.8 %	82.3 %
pretrained VGG11 w/ im. seg.	84.5 %	83.3 %
pretrained VGG16 w/ im. seg.	86.0 %	**84.6** %
pretrained ResNet34 w/ im. seg.	84.4 %	81.6 %

Table 2. Accuracies of single end-to-end CNNs. **im. seg.** means image segmentation (Sec. 3.3) is applied for background removal.

Train-from-scratch models. From Table 2, we can see that the non-pretrained VGG16 performs slightly better than our model implementation. Thus, we proceed with the well-known models only, and see whether using pretrained weights is beneficial.

Without or with pretrained weights. From Table 2, we can see that pretrained VGG16 performs better than a non-pretrained one. In addition, it takes significantly less time to fine-tune pretrained models. It takes 3 to 4 hours to train from scratch, while fine-tuning only takes less than half an hour. Therefore, we consider transfer learning a more efficient way than training from scratch.

Different pretrained models and fine-tuning strategies. After trying different models, we find VGG16 performing the best and also an easier one to train. Besides, we try freezing some of the earlier layers in VGG16 rather than fine-tuning the whole model. Nonetheless, fine-tuning the VGG16 without freezing any layer produces better results.

With or without image segmentation. Results in Table 2 show that training with images having irrelevant background removed leads to higher validation accuracy than with the original images (except in the case of ResNet34). This is probably due to that the model need not learn to focus on the mangoes by itself and that the resolution of the mangoes is higher after resizing.

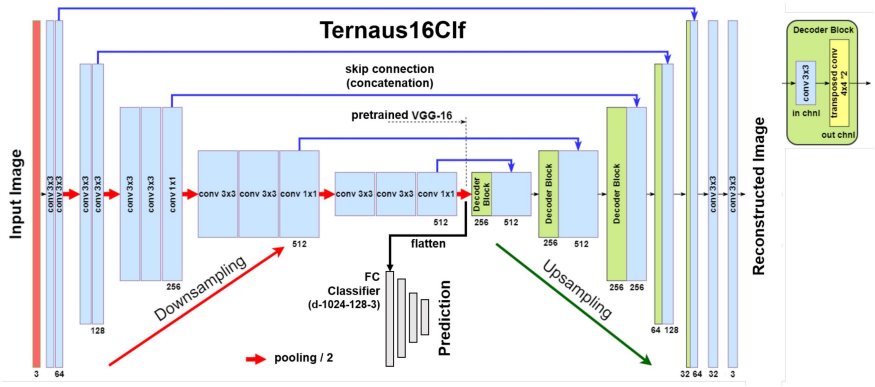


Figure 3. The architecture of **Ternaus16Clf**. (This figure is modified from the TernausNet paper (Shvets et al., ICMLA 2018).)

6.2. Convolutional Autoencoder-Classifiers

Model	Reconstr. loss		Accuracy	
	Train	Val.	Train	Val.
VGG11	*n.a.*	*n.a.*	84.5 %	83.3 %
Ternaus11Clf	.002	.002	92.3 %	84.9 %
VGG16	*n.a.*	*n.a.*	86.0 %	84.6 %
Ternaus16Clf	.004	.004	91.2 %	**85.1** %
ResNet34	*n.a.*	*n.a.*	84.4 %	81.6 %
Albu34Clf	.030	.021	91.5 %	83.6 %

Table 3. Performances of the conv. autoencoder-classifiers, including the direct comparison with their end-to-end CNN counterparts. Reconstruction loss is measured by mean squared error (MSE). All models are initialized with pretrained weights and trained with the data after background removal. Please note that the jump in training accuracy is most likely due to different training settings; therefore, it should not be over-explained.

The reconstruction and classification performances of the convolutional autoencoder-classfiers (ref. Sec. 5) are shown in Table 3. From the validation results we can see that the networks containing VGG encoders (i.e., Ternaus11Clf and Ternaus16Clf), compared to the ResNet34-based Albu34Clf, are not only better autoencoders but also stronger classifiers. Comparing these networks with end-to-end CNNs (see Table 3), we can also observe an incremental performance gain in classification; however, this improvement should be taken with a pinch of salt since our validation data is small (800 pictures) and only few rounds of experiments are run.

6.3. Ensemble Methods and Full Comparison

Ensemble methods combine multiple base classifiers to make a more robust classifier. We use 3 ensemble methods:

- **hard voting**—simple majority vote by the classifiers;
- **soft voting**—making predictions from the avg. of the classfiers' softmax-ed probabilities;
- **logistic regression**—training another classifier using the classifiers' softmax-ed probabilities as features.

The complete results of all models are shown in Table 4. Concerning stand-alone models, VGG16 and Ternaus16Clf are the best performers, both achieving a test set accuracy of just over 80%. For ensemble methods, while all of them give an extra boost on the accuracy, hard voting performs the best (81.8% test acc.); however, we find that, rather than the ensemble technique, the base classifiers involved in the ensemble are the determining factor of its performance, hence should be carefully chosen for each ensemble method. Moreover, the performance on the val. set doesn't translate directly to the test set; this could be because the val. set accuracy is closely monitored during training and used as a model selection criterion, thus leading to implicit overfitting.

Grp.	Model	Accuracy (%)		
		Train	Val.	Test
(a)	**baseline mod.** (9.4M)[1]	96.7	74.5	73.3
(b)	**our own CNN** (28.0M)	81.3	80.1	*n.a.*
	AlexNet (57.0M)	81.8	82.3	*n.a.*
	VGG11 (128.8M)	84.5	83.3	*n.a.*
	VGG16 (143.3M)	86.0	84.6	**80.1**
	ResNet34 (21.3M)	86.7	82.4	79.9
(c)	**Ternaus11Clf** (58.0M)	92.3	84.9	77.9
	Ternaus16Clf (72.7M)	91.2	85.1	**80.3**
	Albu34Clf (63.8M)	91.5	83.6	79.0
(d)	**ensb.—hard voting**[2]	*n.a.*	85.0	**81.8**
	ensb.—soft voting[3]	*n.a.*	84.1	80.8
	ensb.—logistic reg.[2]	*n.a.*	87.5[4]	81.1

[1] () contains the total # trainable parameters of the model.
[2] Ternaus16Clf, VGG16, and ResNet34 are included.
[3] Ternaus11Clf, Ternaus16Clf, VGG16, ResNet34 are included.
[4] Trained on val. set w/ 5-fold cross-validation; param C=0.4.

Table 4. Comparison of all models involved in this work. All models, except **baseline model** and **our own CNN**, are fine-tuned on the basis of pretrained weights and uses the data after background removal. The end-to-end CNNs (group **(b)**) and the conv. autoencoder-classifiers (group **(c)**) perform comparably and much better than the baseline. Ensemble methods (group **(d)**) also give incremental performance gains.

6.4. On the Misclassified Samples

Since VGG16 and Ternaus16Clf (which contains VGG16 encoder) outperform other models in our experiments, we decide to take a closer look into the VGG16's predictions.

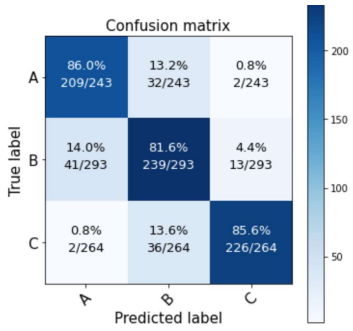
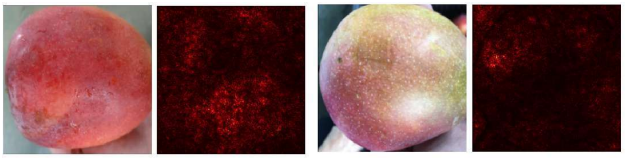
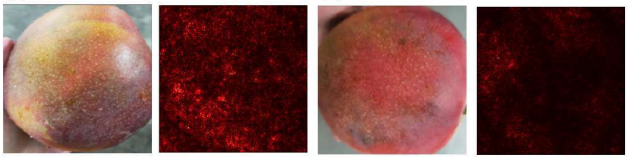


Figure 4. The confusion matrix of the pretrained VGG16 on the validation set.

Confusion matrix. From Figure 4, we can see it is harder for the model to tell apart class A and class B mangoes. Also, class C mangoes are often misclassified as class B.



(a) True label: B, Predict label: A (b) True label: C, Predict label: B



(d) True label: A, Predict label: B (d) True label: A, Predict label: B

Figure 5. Misclassified samples and their corresponding saliency maps. The saliency maps (Simonyan et al., arXiv 2014) next to each samples visualize the attention of VGG16. Red pixels in the saliency maps indicate large gradients w.r.t. the predicted classes.

Case study. We sort the misclassified mangoes by their loss values. Figure 5 presents some of the samples with higher loss. From Figure 5, we can see the model puts most of its attention on the mangoes and defects like black dots. However, Figure 5(c) suggests that the model is still affected by the noisy background. To remedy this, we may try fine-tuning the Mask R-CNN with more samples to get better background removal results.

Looking at the misclassfied images, we find the model makes wrong predictions on samples involving uneven skin color and rarer diseases more often. Moreover, we find the labeling standard quite inconsistent. For instance, some mangoes with uneven color are labeled as A, while some are labeled as B. On the discussion board, the competition organizer also admit that the dataset is labeled by several judges, and that each sample is annotated by one judge; hence, we consider the quality of human annotation questionable.

7. Conclusion

In this work, we investigated several deep learning-based methods to approach the mango grading problem. Through our experiments, we discovered that removing the irrelevant background of images and making use of pretrained models are the most effective ways to boost the accuracy. Furthermore, the convolutional autoencoder-classifiers we proposed were shown to perform at least as well as the well-known CNNs on the public test set, or slightly better on the validation set. Nevertheless, its competence is still yet to be verified with larger datasets and more diverse tasks.

8. Teammate Responsibilities

Our teammates are responsible for the implementation, experiments, and the writing of:

- **B06401036 Hsiao-Yen Tung**: Sections 4 and 6.
- **B06401078 Yu-Lun Hsu**: Sections 3 and 6.
- **B06902080 Shih-Lun Wu**: Sections 1, 2, 5, 6, and 7.

References

Canny, J. A computational approach to edge detection. PAMI 1986.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. ACM SIGKDD 2016.

Dutta, A. and Zisserman, A. The via annotation software for images, audio and video. ACM MM 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. CVPR 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. CVPR 2016.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. Mask r-cnn. CVPR 2017.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. NIPS 2012.

Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. NIPS 2015.

Shvets, A. A., Iglovikov, V. I., Rakhlin, A., and Kalinin, A. A. Angiodysplasia detection and localization using deep convolutional neural networks. ICMLA 2018.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. ICLR 2015.

Simonyan, K., Vedaldi, A., and Zisserman, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv 2014.