Comparing 3 Classifier Models

Candidate no 4

***Abstract*—The objective of this report is to compare the effectiveness and robustness of three pre-trained PyTorch classification models, as well as see how much better the models perform when starting with pre-trained weights as opposed to being initialized randomly before training. In the report 3 experiments are done where the success parameter is accurate classification of aerial landscape photos into 15 categories. The results of experiment 1 indicate that initializing with pre-trained weights lead to better results much faster than initializing with random weights, and approximately 20% better performance in reasonably short training time. The results of experiment 2 suggests that the implementation of data augmentation should be done in a way more subjective to each model, and be tuned, to get good results. In the final experiment ResNet displays superior robustness for salt and pepper noise, while AlexNet was shown to be more robust for gaussian noise.**

***Index Terms*—Deep Learning, supervised learning, SGD**

I. Introduction

In this report three different Pre-trained PyTorch classification models are trained, tested and compared for effectiveness and robustness. The models chosen are AlexNet, VGG and ResNet. The report describes where they come from and how they are built up. The testing of the models is conducted on a dataset containing aerial landscape photos divided into 15 categories. The report explores how well the models work with the best available pre-trained weights for each of them versus being initialized with random weights. Furthermore an experiment is done to see if the test performance is better when the data the models are trained on is augmented with colour jitter and cutout. Finally the robustness of the models is examined by comparing the test performance with no noise, Gaussian noise and salt and pepper noise.

II. Theory

In this section the general theory behind Convolutional Neural Networks and the specific CNN models chosen in this report is presented. The models chosen for this report in specific is AlexNet, VGG-16 and ResNet-34. The information about these models are primarily sourced from their original papers, which are "ImageNet Classification with Deep Convolutional Neural Networks" [1], "Very Deep Convolutional Networks for Large-Scale Image Recognition" [2] and "Deep Residual Learning for Image Recognition" [3]. Information about neural networks and their design is mainly sourced from the book "Alice's Adventures in a differentiable wonderland" [4].

A. Convolutional Neural Networks (CNN)

A CNN is a type of artificial neural network (ANN). An ANN is a type of computational model that is inspired by and loosely modelled after the neurons and structure in a brain, and is designed to recognize patterns in data. CNNs are designed to work well with spatial data, for example in the scope of this report, image data. They are built up of multiple layers, that each take the previous layer as an input and has its own set of trainable parameters. The most important layers of these networks are Fully Connected layers (FCL) and Convolutional layers (CL).

A CL works by using a filter/kernel of a given dimension that "slides" over the image and can capture patterns in it. Some key hyper-parameters are for this layer are:

- Filter size: e.g. 3x3, 5x5.
- Stride: How many pixels the centre of the filter moves per sample
- Number of Filters: These each learn a distinct feature.

These layers are often followed by pooling layers, like Max-pooling and Global average pooling (GAP), that are used to downsample the output of the CL, which is to reduce its spatial dimensions (height and width) and computational complexity.

Max-pooling samples the output in a way similar to CLs with a filter size and a stride, but the channel number stays the same while the spatial dimensions decrease, as shown in figure 1. This figure also shows how the max-pooling function returns the maximum value in its filter for each step, which helps to conserve the most important features in each region while reducing size.

In contrast when using GAP, no spatial information is conserved. The function returns a single vector containing the average of all values in each channel, effectively collapsing the output to one dimension. As can be seen in ResNet-34 the GAP layer can be used to replace the final FCLs used in other models as the final classification layer.

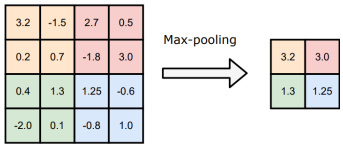

Fig. 1. Visualization of 2x2 maxpooling on a (4,4,1) image. For multiple channels, the operation is applied independently on each channel [4].

In FCLs all neurons in the layer are connected to the layer before. In convolutional networks these are usually implemented as some of the last layers, for example to aggregate features for a classifier. To reduce overfitting the model to the training data it is not unusual to implement dropout on these final layers, which essentially cancels out a set amount (often half) of the neurons randomly.

To avoid the layers of the network collapsing into linear transformations it is necessary to use an activation function to

introduce some form of element-wise non-linearity between layers. The choice of this non-linearity is a big influence on the gradient, and choosing a function that is non-linear enough to prevent collapse, but still stays close enough to its derivative identity. A good default choice for such a function is the Rectified Linear Unit (ReLU), which was popularized originally by AlexNet, in replacement of the convention of using tanh or the sigmoid function.

B. AlexNet

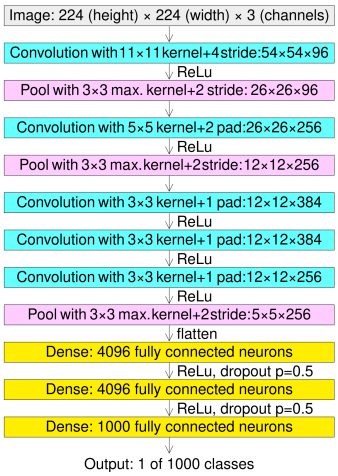

Fig. 2. Overview of AlexNet architecture [5]. (Original image cropped to exclude irrelevant information).

This CNN is named after its creator Alex Krizhevsky and was created for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2010/2012. It consists of 5 CLs, 3 FCLs, 3 max-pooling layers, uses ReLU for introducing non-linearity and uses dropout to reduce overfitting, as shown in figure 2. A form of local response normalization is also applied after the two first CLs, which the authors found to reduce the models error rates.

Some key points in AlexNet that makes it stand out from other implementations from around the same time:

- One of the first models to use ReLU instead of tanh/sigmoid, which significantly improved training times.
- Utilizes Local Response Normalization as a form of generalization after some layers, which has been shown to reduce the test error rate. In their paper it is termed a "brightness normalization".
- Popularized using dropout in the FCLs to reduce overfitting.
- Parameter count $\approx$ 62 million.

The implementation of AlexNet used in this report is implemented in the PyTorch package based on the version of ALexNet presented in the paper "One weird trick for parallelizing convolutional neural networks" [6]. The hyper-parameters this model was initially trained on are:

- Initial learning rate = 0.01, reduced manually by a factor of 10 when the validation error plateaued.
- Weight Decay = 0.0005 (L2 regularization)
- Batch Size = 128
- Epochs = 90
- Optimizer = Stochastic Gradient Descent (SGD) with Momentum = 0.9
- Dropout Rate = 0.5
- Data augmentation:
  - Random cropping
  - Horizontal flipping
  - RGB intensity shifting

C. VGGNet

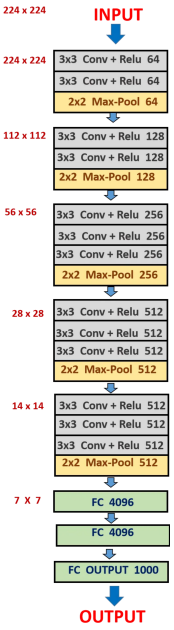

Fig. 3. Overview of VGG-16 architecture [7].

This CNN was made by Visual Geometry Group (hence the name) at the university of Oxford for ILSVRC-2014. VGG-16, which is one of the more commonly used versions consists of 13 CLs and 3 FCLs. Its architecture, as can be seen in figure 3, is similar to that of AlexNet, but utilizes a deeper network with more CLs, all using a smaller filter size of 3x3. Also to be noted is that they did not find AlexNets local response normalization to improve performance on this model, only increase memory and time consumption with no added benefits.

Some key points in VGG-16:

- Utilizes a smaller filter size (3x3) for CLs that reduces the amount of parameters while enabling the model to learn more complex features.
- Easily understandable and reproducible architecture due to all CLs being 3x3, stride = 1.
- Inspired Modular/Scalable design
- Highlights the inefficiency of high parameter counts with $\approx$ 138 million parameters.

The hyper-parameters this model was initially trained on are:

- Initial learning rate = 0.01, reduced by a factor of 10 when validation accuracy plateaued.
- Weigth Decay = 0.0005 (L2 regularization)
- Batch Size = 256
- Epochs = 74
- Optimizer = Stochastic Gradient Descent (SGD) with Momentum = 0.9
- Dropout Rate = 0.5
  - Random cropping
  - Horizontal flipping
  - RGB Colour shifting

D. ResNet

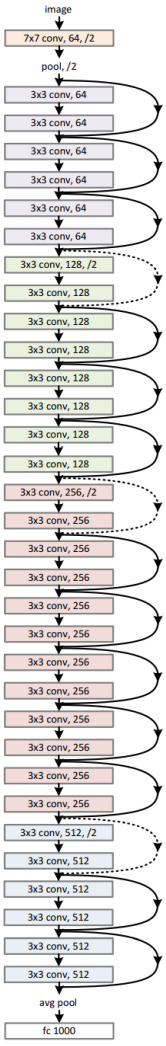

Fig. 4. Overview of ResNet-34 layers [8]. (Original image cropped to exclude other models).

This CNN was made by Microsoft Research for ILSVRC-2015 as part of the ResNet family, which won the competition. The model is known for introducing residual connections, which enables training very deep network architectures without having vanishind gradient issues (which is when the gradient diminishes to essentially zero). As shown in figure 4, ResNet similarly to VGG utilizes almost exclusively 3x3 CLs, but instead of FCLs as end layers they implement GAP, which gives it a decent reduction in parameter count.

The main feature of this model is the residual connections (also called skip-connections). These work by adding the layers input to its output like in equation 1:

$$Output = f(x) + x \tag{1}$$

where x is the input.

This makes it so the model only has to learn from the residuals, and not the entire transformations at each layer, which is easier. In turn this allows the model to learn ”identity mappings” easier, which allows it to pass input through unchanged, essentially skipping the layer. This reduces the complexity of what each layer must learn.

Some key points in ResNet-34:

- Utilizing residual connections to allow for training deeper networks without performance loss or vanishing gradients.
- FCLs replaced by GAP to reduce parameter count, also eliminates dropout.
- Relatively low parameter count $\approx$ 21.8 million.

The hyper-parameters this model was initially trained on are:

- Initial learning rate = 0.01, reduced by a factor of 10 at specific epochs (e.g., 30 and 60 in a 90-epoch schedule).
- Weight Decay = 0.0001 (L2 regularization)
- Batch Size = 256
- Epochs = 90
- Optimizer = Stochastic Gradient Descent (SGD) with Momentum = 0.9
- Data augmentation:
  - Random cropping
  - Horizontal flipping

E. Data

The dataset used in this report consists of 12 000 aerial landscape images over 15 categories:

- Agriculture
- Airport
- Beach
- City
- Desert
- Forest
- Grassland
- Highway
- Lake
- Mountain
- Parking
- Port
- Railway
- Residential
- River

The dataset is randomly divided into training, validation and testing dataset with sizes 8400, 1800, 1800. 4 example images are shown in figure 5:

Fig. 5. Example images with labels: Highway, Railway, Port, Parking.

F. Data Augmentation

Two methods of data augmentation are used for the training and validation sets after the first experiment: colour jitter and cutout. Colour jitter is done by randomly changing colour aspects of the image like brightness, contrast, saturation and hue to simulate a high range of time of day, lighting and weather conditions, which helps with generalization. Cutout randomly ”cuts” out a part of the image of a given size by making the pixels there black. This can simulate occlusion, like if a tree was covering a road, and it can encourage the model to learn to watch for larger features in the image instead of focusing on smaller details. 4 example images augmented by both methods are shown in figure 6:

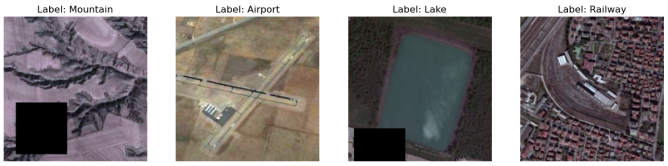

Fig. 6. Augmented example images (cutout has an x% chance of activating).

For augmentation of test images to test robustness two types of noise are tested: Gaussian and salt and pepper. Gaussian noise is random noise with a normal distribution added to the pixel values to simulate sensor noise or other environmental distortions that may occur in photos. When used on training dataset this augmentation is known to increase robustness of the model. Salt and pepper noise makes random pixels appear black or white with a chosen probability.

G. Performance metrics

The main metrics used to describe the models performance are loss and accuracy. Loss can be explained as how ”unsure” the model is about its guesses, and is calculated with a loss function. When working with a classifier it is normal to use Cross-entropy loss, which is the difference between the guess and the ground truth. This metric is important as this is what the optimizer aims to reduce. Accuracy is an absolute measurement of how many predictions are actually correct. This is the metric mainly being observed in evaluation, as the most important metric should be how accurate the model is in practice.

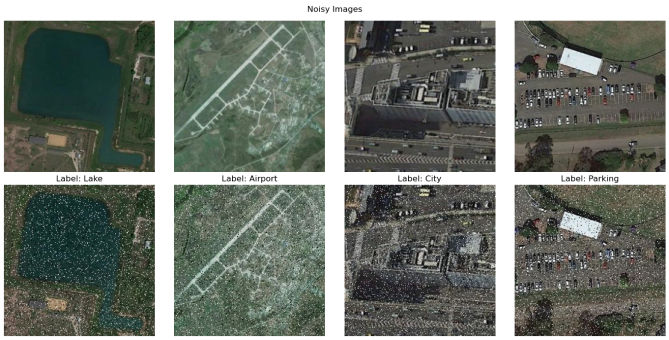

Fig. 7. Top row shows images with added gaussian noise with std = 0.01, bottom row shows images with salt and pepper noise with probability of 0.05 for each. (Gaussian noise with low standard deviation is not necessarily expected to be seen

III. Methodology

The models to be used were chosen to see how progression in the development in CNNs has progressed over time from AlexNet, which was a breakthrough model that acted as a catalyst for deep learning progression due to reducing the error rate by a significant margin, to ResNet, that introduced concepts that are still used in state of the art networks. For all three models, the experiments, hyper-parameters and optimizer used during training etc. are the same to make evaluating the performance of the different models as equal as possible.

To prepare the images for training they are resized from the original 256x256 to 224x224 to fit the models input, and each channel is normalized with mean = [0.3781, 0.3930, 0.3445] and std = [0.1353, 0.1230, 0.1177] calculated from the dataset.

For training the hyper-parameters used for all three models are the same. These are:

- Batch size = 64 for first experiment, 32 onwards because of memory usage
- Learning rate = 0.0001, because these models are pre-trained
- Weight decay = 0.0005, this is taken directly from AlexNet paper, and seems to work so it was not changed
- Momentum = 0.9, Also taken from AlexNet
- Optimizer = Adam, This was chosen as it is said to be relatively robust in the choice of its hyper-parameters, and the default choice working well in most cases [4].
- First experiment epochs = 15, for brevity
- Epochs for following experiments = 30, but now including early stopping, so this is not reached.
- Early stopping patience = 5 and delta = 0.0001

The first experiment is about training 2 versions of each model, one with the best currently available weights and one initialized with random weights. The loss and accuracy over time for the 2 versions during training is compared and the final performance of all six trained models on the test set is compared. In this experiment the models were trained for 15 epochs, as that should be enough to see if it is preferential to initialize with pre-trained weights.

For the next experiment the training and validation data is augmented with colour jitter and cutout, the optimizer gets L2 weight regularization and the training process is updated with early stopping. In this experiment the three models performance with these processes applied is compared to the performance of the same model in the first experiment to see if these augmentations facilitated improvement in training.

The final test aims to test the models robustness by adding noise to the test dataset. I found two types of noise to be interesting to test, so I implemented gaussian noise and salt and pepper noise separately and compared the result to performance on the same dataset with no noise.

IV. Results

A. Experiment 1

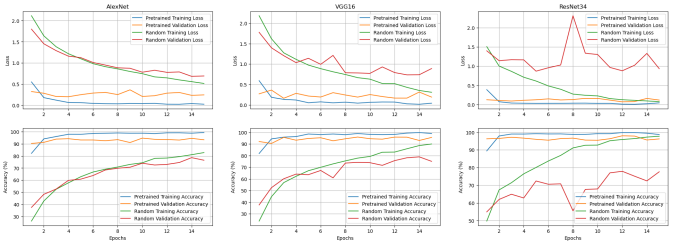

Fig. 8. Model performance during training, divided by metric and model.

From figure 8 we can see that for all three models, the version that started with pre-trained weights have much higher accuracy from the start. The randomly initiated versions reach low loss levels, but their accuracy is not good enough. The models with pre-trained weights output high accuracy quite fast, while their counterparts with random weights perform approximately 20% worse on the test set, as seen in figure 9.

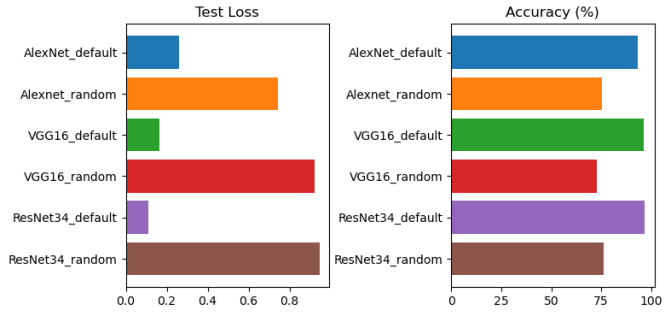

Fig. 9. Test loss and accuracy for both variations of all models.

B. Experiment 2

As we can see from table I, with the changes made after the first experiment the AlexNet models Loss is reduced by 0.03 and accuracy increases by almost 1%, but the two other models perform worse, with VGG-16 losing more than 1% accuracy.

Model	Loss	Accuracy (%)	Augmented
AlexNet	0.26	93.00	no
AlexNet	0.23	93.94	yes
VGG-16	0.17	96.33	no
VGG-16	0.16	95.00	yes
ResNet-34	0.11	96.72	no
ResNet-34	0.12	96.44	yes

TABLE I
RESULTS OF TEST SET EVALUATION OF MODELS TRAINED ON NORMAL AND AUGMENTED DATASET.

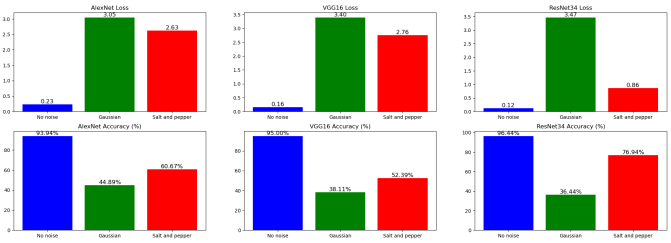

Fig. 10. Loss and accuracy of the models compared for no noise, Gaussian noise and salt and pepper noise.

C. Experiment 3

We can see from figure 10 that for normal no-noise test data, the newer the model, the better the performance. This goes for both accuracy and loss. For the test data with gaussian noise added it seems this relationship is reversed, where AlexNet now performs the best. For the last test with salt and pepper noise ResnNet is back on top with accuracy more than 16% better than AlexNet and 24% better than VGG. When we look at loss we can see that ResNet achieves less than a third of the other models loss for this type of noise.

Link for anonymized GitHub page containing the Jupyter notebook with the python implementation of concepts used in this report: https://anonymous.4open.science/r/MachineVisionExam. This notebook was not made to be pretty or to be reusable, so it is not neatly commented or documented, but serves to show how the experiments were done in practice, should one be interested.

V. Discussion

A. Experiment 1

When trained on an image dataset already the model has already learned to identify some generalized features and shapes and it is then easier and more effective to train for a new dataset than if initialized with random values. I also discovered that I had to reduce the batch size from 256 (which worked fine with AlexNet) to 64 to be able to train the VGG-16 model. This shows that a deeper model with more parameters might increase memory usage to a level that impedes the training speed. This result aligns well with the expected outcome of this experiment.

B. Experiment 2

This experiment did not really yield the results I was thinking I might see. The data augmentation was thought to make the model more robust, and maybe train the model

more effectively, but instead, the newer models got worse performance and only AlexNet profited from it. I think I might have implemented the colour jitter poorly, and also due to long training time I have not tested different values for the augmentation. So in essence the parameters for the augmentations were more of an educated guess and crossed fingers for good performance, in all honesty. Upon later reflection I would maybe have used one of the models with lower training time as a reference and tried to discover the best practices for data augmentation before training VGG-16, which took a long time. This would include exploring other types of augmentations and changing the parameters.

C. Experiment 3

For the no-noise performance, it is to be expected that newer models with more advanced architecture performs better, as is shown. For the gaussian noise I am suspecting that due to AlexNet being a simpler network it does not rely as much on the subtle patterns in the images, as it learns more low-level features. One could perhaps think of the other models overfitting to the ”clean” images due to their more complex nature. For the salt and pepper noise it makes sense that ResNet performs better, because of its skip-connections it can effectively ignore the ”corrupted” pixels and rather focus on the continuity of features across multiple layers.

D. Generally

After completion of this report I think the things I would have done differently would be to try to train the model on data augmented with gaussian noise instead of colour jitter, and maybe to test more different values for hyper-parameters in general and for augmentation and noise. Due to time constraints and general workload over all subjects during this exam period I do not at time of writing this have enough time left to run more training or implement more features. Even though we were given about a month of time for this report I was nowhere near able to start that early with this report because of the rest of my course-load.

VI. Conclusion

A. Experiment 1

This conclusion is quite easy: using a pre-trained model allows you to in a way skip time in training as it already has learned a generalized form of image recognition and classification that only needs to be fit to the new dataset.

B. Experiment 2

For this experiment I will conclude with the fact that data augmentation can benefit a models training as long as it is done correctly, and that the type of augmentation should be both relevant for the dataset and be of a type that fits the architecture of the model. In essence, the type of augmentation should maybe be chosen more specifically for each model.

C. Experiment 3

I would say the most robust model of these three is ResNet-34, as it performs the best overall. The differences in accuracy for the gaussian noise is half that of the difference in performance for the salt and pepper noise, where it also had much less loss than the other models. Summed up, more complex models seem to perform well with clean data and with structured noise like salt and pepper noise, while simpler models may handle unstructured noise like gaussian noise better due to not relying as heavily on the images subtle patters.

Lastly I would say that I have learned a lot during the making of this report, and had I started over with the knowledge I have now I am confident the implementation would have been cleaner and more thought through. Also I would do more extensive testing of parameters and methods used to determine the best ones possible.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” Ph.D. dissertation, University of Toronto, 2012.

[2] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” Department of Engineering Science, University of Oxford, Tech. Rep., 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” Microsoft Research, Tech. Rep., 2015.

[4] S. Scardapane, “Alice’s adventures in a differentiable wonderland – volume i, a tour of the land,” 2024.

[5] Cmglee, “Comparison image neural networks.svg,” 2021, [Accessed 15-November-2024]. [Online]. Available: https://commons.wikimedia.org/wiki/File:Comparison_image_neural_networks.svg

[6] A. Krizhevsky, “One weird trick for parallelizing convolutional neural networks,” Google Inc., Tech. Rep., 2014.

[7] V. Khandelwal, “Vgg-16,” 2020, [Accessed 16-November-2024]. [Online]. Available: https://pub.towardsai.net/the-architecture-and-implementation-of-vgg-16-b050e5a5920b

[8] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, “Vgg module architecture.svg,” 2023, [Accessed 16-November-2024]. [Online]. Available: https://commons.wikimedia.org/wiki/File:VGG_module_architecture.svg