MobileNets: Efficient ConvNet for Mobile Vision Applications

Yatharth Bansal, Ajinkeya Chitrey, and Aishwarya Patange

Department of Electrical Engineering
Columbia University
New York, NY

{*yb2540, ac5166, aap2239*}@columbia.edu

***Abstract*—In this project, we study and implement an efficient and simple convolutional neural network architecture called as MobileNets which can be used for mobile and embedded vision applications mentioned in the paper "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". The proposed architecture in the paper is a compact efficient deep neural network built incorporating and leveraging depth-wise separable convolutions. The paper showcases a solution for dealing with the trade-off between latency and accuracy by proposing the usage of two new global hyperparameters. With the help of these hyperparameters, the model gains the advantage to chose the required sized model for a given application to offer lower latency. We showcase the significant and efficient performance of the model in terms of lower latency and higher accuracy as described by the authors in the paper by conducting extensive experimentation and providing performance comparisons to other models based on ImageNet classification. We depict the models' implementation and significant results in various mobile and embedded vision applications like fine-grained recognition, face attribute classification, object detection, and face embeddings.**

***Index Terms*—MobileNets, CNN, depth-wise separable convolutions, object detection**

I. Introduction

Convolutional neural networks (CNN) has recently become a crucial, key, and imperative component for computer vision. In recent years, usage and adaptation of CNNs to computer vision tasks have seen tremendous growth. The trends in the experimentation as observed in [1]–[3] based on CNNs for these tasks depict an intent to increase the depth of the network and build more complex networks for achieving better performance. But with the increase in network depth and complexity of networks, a trade-off in latency can be observed compromising the speed and size of the network. Implementation of these complex architectures in real-world applications becomes a difficult and impractical task where the expectations are to serve high speed and accuracy on limited computational platforms.

In the recent literature, [4]–[6], an increasing interest can be observed to build neural network architectures which are small and efficient to overcome the issues discussed previously. Among the various different approaches discussed in the recent literature, the methods for modeling a smaller efficient network mainly consist of either direct training of small networks or squeezing the pre-trained networks. Even with the intent to build smaller architectures, many of these recent works fail to justify the speed and efficiency of these neural networks to be deployed in real-time applications.

In this project, to navigate and deal with the issues discussed, we study and implement an efficient and streamlined architecture named as MobileNets proposed in [7]. In this work, the author proposes an efficient neural network architecture along with a set of two hyperparameters that allows building custom application-specific simple models offering low latency and higher accuracy in performance. Depth-wise separable convolutions introduced in [8] form the building blocks of the MobileNets making it capable of reducing its computations. MobileNets efficient and small architecture serves to demands of mobile and vision-embedded applications.

The authors in [7], depict the performance of MobileNets by experimenting and comparing their model to the well-known popular models to showcase the resource and accuracy tradeoffs and significant performance of MobielNets. In this project, we corroborate the performance of MobileNets by performing extensive experimentation and showcasing its performance to other models based on ImageNet classification. We also implement MobileNets for various mobile and vision-embedded applications like fine-grained recognition, face attribute classification, object detection, and face embedding showcasing the significant performance achieved for each application-based task.

The paper has been divided into five sections. In section II, we describe the methodology of this work and describe the architecture in brief. The implementation of the work, experiments conducted and results are described in section III. Finally, in section IV, we conclude our work and discuss the potential future work that can be done in this project.

II. Literature Review

In this section, we describe the architecture of the MobileNets and lay out the key components contributing towards efficient performance. Firstly, we briefly explain the core of MobileNets which builds the bases of architecture, i.e., depth-wise separable convolution. Secondly, we elaborate on the architecture and define the set of two hyperparameters that are responsible for model shrinking, i.e, width multiplier and resolution multiplier.

At the heart of this paper lies the concept of factorized convolutions. In standard convolutions, for each output element, we sum the product of the convolutional kernel and the corresponding input elements. In other words, they try to get output by convolving a 3-dimensional kernel with the input feature along their spatial dimensions. The calculation complexity for this task can be given by multiplying the size of the input, the size of the kernels, and the number of channels of the inputs.

The authors in the paper try to effectively achieve the same convolutional output and reduce the number of mathematical operations at the same time by using factorized convolutions. Factorized convolutions try to recreate the original standard convolutional kernels by decomposing them into two smaller kernels. This paper uses the depthwise separable factorization technique. Depthwise separability is nothing but converting the 3-dimensional standard convolutions into 2-dimensional spatial convolutions and then taking a 1-dimensional linear projection across the channels. The spatial convolution effectively captures the local structural information while the linear projection introduces non-linearities in the feature space. We can also look at the spatial convolutions to have a filtering effect while the linear projections to have a combining effect. This technique can also be referred to as single intra-channel convolutional layer followed by a linear projection [5].

To understand why factorized convolutions are more efficient than standard convolutions in terms of computational complexity, we need to look at the number of mathematical operations that take place in each of the two techniques. If a standard convolutional layer takes in an input feature map $\boldsymbol{A}$ having the dimensions $D_{A_1} \times D_{A_2} \times M$ to produce an output $\boldsymbol{B}$ having the dimensions $D_{B_1} \times D_{B_2} \times N$ using convolutional kernels $\boldsymbol{K}$ having the dimensions $D_{K_1} \times D_{K_2} \times M \times N$, the total computational cost of it would be:

$$D_{K_1} \times D_{K_2} \times M \times N \times D_{A_1} \times D_{A_2}$$

For the depthwise separable convolution, we analyse the computational cost of the depthwise convolutions (2-dimensional spatial convolutions) and the pointwise convolutions (linear projections). The depthwise convolutions work with one filter per input channel. Hence, if we take a kernel $\hat{\boldsymbol{K}}$ having the dimensions $D_{K_1} \times D_{K_2} \times 1$ and apply it on the m_{th} channel $\boldsymbol{A_m}$ having the dimensions $D_{A_1} \times D_{A_2} \times 1$ of the input feature map $\boldsymbol{A}$ to produce the m_{th} channel A_m of the output feature map $\hat{\boldsymbol{B}}$ with dimensions $D_{B_1} \times D_{B_2} \times 1$, the total computation cost will be:

$$D_{K_1} \times D_{K_2} \times M \times D_{A_1} \times D_{A_2}$$

This output feature map $\hat{\boldsymbol{G}}$ is now fed to the combination layer which is nothing but $1 \times 1 \times M$ kernel which combines individual elements of all the M channels to get a single channel output feature map. This linear combination is done N times to get an output feature map of N channels. Hence, the total computational cost required by the whole depthwise separable convolution technique is:

$$(D_{K_1} \times D_{K_2} \times M \times D_{A_1} \times D_{A_2}) + (M \times N \times D_{A_1} \times D_{A_2})$$

If we try to get the ratio of the computational cost of the depthwise separable convolutional operation to the computation cost of the standard convolutional operation, we find out that the depthwise separable convolutions take $\frac{1}{N} + \frac{1}{D_{K_1} \times D_{K_2}}$ times the computations required by a standard convolution operation. If we take a convolutional filter of dimensions 3×3, the depthwise separable convolution will be almost 9 times faster.

Further, we implemented this model using the Keras API of the tensorflow library. The first thing we did was create separate functions for the factorized convolutions. The first function consisted of us creating depthwise convolutional layer. This layer consisted of the `DepthwiseConv2D` layer followed by a `BatchNormalization` and a `ReLU` layer. The second function was that of the pointwise convolutional block. This involved a `Conv2D` layer coupled with a `BatchNormalization` and a `ReLU` layer. We also give the users the option of adding a `Dropout` layer after the activation layer.

Since we follow the original paper, we start off by model by an iteration of the standard convolutions layer which include `Conv2D`, `BatchNormalization`, `ReLU` and the `Dropout` layers. After this, we start with the first `DepthwiseSeparableBlocks` mentioned earlier. We keep stacking these until we have 13 of these blocks and we successively keep increasing the number of filters after every 2 blocks. Once our `DepthwiseSeparableBlocks` convolutions are over, we use the `GlobalAveragePool` layer and then flatten the outputs. This output is then sent to a `Dense` layer which has a $sigmoid$ or a $softmax$ activation function depending on the number of classes we need to predict. Table 1 gives us the architecture of the network used by the authors of the original paper. We implemented an identical structure with changes in the input and the output layers depending on the experiment we performed.

TABLE I
MOBILENET ARCHITECTURE

Type / Stride	Filter Shape	Input Size
Conv / s2	3 * 3 * 3 * 32	224 * 224 * 3
Conv dw / s1	3 * 3 * 32 dw	112 * 112 *32
Conv / s1	1 * 1 * 32 * 64	112 * 112 * 32
Conv dw / s2	3 * 3 * 64 dw	112 * 112 * 64
Conv / s1	1 * 1 * 64 * 128	56 * 56 * 64
Conv dw / s2	3 * 3 * 128 dw	56 * 56 * 128
Conv / s1	1 * 1 * 128 * 128	56 * 56 * 128
Conv dw / s2	3 * 3 * 128 dw	56 * 56 * 128
Conv / s1	1 * 1 * 128 * 256	28 * 28 * 128
Conv dw / s2	3 * 3 * 256 dw	28 * 28 * 256
Conv / s1	1 * 1 * 256 * 256	28 * 28 * 256
Conv dw / s2	3 * 3 * 256 dw	28 * 28 * 256
Conv / s1	1 * 1 * 256 * 512	14 * 14 * 256
5 * Conv dw / s1	3 * 3 * 512 dw	14 * 14 * 512
5 * Conv / s1	1 * 1 * 512 * 512	14 * 14 * 512
Conv dw / s2	3 * 3 * 512 dw	14 * 14 * 512
Conv / s1	1 * 1 * 512 * 1024	7 * 7 * 512
Conv dw / s2	3 * 3 * 1024 dw	7 * 7 * 1024
Conv / s1	1 * 1 * 1024 * 1024	7 * 7 * 1024
Avg Pool / s1	Pool 7 * 7	7 * 7 * 1024
FC / s1	1024 * 1000	1 * 1 * 1024
Softmax / s1	Classifier	1 * 1 * 1000

TABLE II
EFFECT OF THE WIDTH PARAMETER α

Paper Architecture	Paper Accuracy (ImageNet)	Implemented Architecture	Implemented Accuracy (CIFAR-10)
1.0 MobileNet-224	70.6%	1.0 MobileNet-32	61.59%
0.75 MobileNet-224	68.4%	0.75 MobileNet-32	61.38%
0.5 MobileNet-224	63.7%	0.5 MobileNet-32	51.20%
0.25 MobileNet-224	50.6%	0.25 MobileNet-32	40.22%

TABLE III
MOBILENET FOR FINE GRAINED RECOGNITION

MODEL	Accuracy obtained (Trained on a subset of 40 classes of Stanford dataset)	Accuracy reported in the paper (Trained on all 120 classes of Stanford Dataset)
1.0 MobileNet-224	84.8%	83.3%
0.75 MobileNet-224	82.32%	81.9%
1.0 MobileNet-192	81.73%	81.9%
0.75 MobileNet-192	81.2%	80.5%

III. IMPLEMENTATION

A. Experimentation

1) Model Shrinking Hyperparameters: In this experiment we try to show the effect of changing the α parameter in the MobileNet architecture. The α parameter is also called as the width multiplier which basically thins the network uniformly at each layer, creating an effectively thinner network. We trained the CIFAR-10 dataset [9] on a custom implementation of the MobileNet architecture. The input images were of size 32×32. We trained the model for 10 epochs for $\alpha \in [1, 0.75, 0.5, 0.25]$. In the original paper, the authors trained their models on the ImageNet dataset with an input image size of 224×224 pixels. As we can see in Fig. 1, decreasing the alpha value reduced the accuracy of the model but at the same time decreased the amount of time taken to train each epoch. Hence there is a tradeoff between accuracy and size of the network. We do a comparison between the papers results and the results obtained by us by changing the width parameter in II. We observed a similar trend of the accuracy decreasing with the decrease in the value of α
. Total training time per epoch:

1) $\alpha = 1 \Rightarrow 630s$
2) $\alpha = 0.75 \Rightarrow 350s$
3) $\alpha = 0.5 \Rightarrow 174s$
4) $\alpha = 0.25 \Rightarrow 93s$

Total training time over all model iterations ≈ 3.5 hours

2) Fine-Grained Recognition: In this experiment, we evaluate the performance of MobileNet on a noisy dataset that requires fine-grained recognition capabilities. For this we downloaded the Stanford Dogs dataset as done in the original experiment in [7]. We subsampled 40 classes of dog images from the overall 120 to respect computational constraints. In addition to this we created synthetic data by using techniques such as rescaling, shearing, zooming, flipping, and rotating. MobileNet model Keras API was used for using the model for experimentation. An additional global average pooling layer and a Softmax dense output layer with 40 neurons were added to it. The final model was trained on the data for different width multiplier alpha and resolution values as done in [7]. The results obtained have been showcased in Table III and Fig 2.

Training time per epoch on average = 170 seconds

Total training time over all model iterations = 3 hours

3) Face Attributes: In this experiment, we use MobileNet to efficiently identify the defining facial attributes of people's faces. Due to computational constraints, to recreate this experiment, instead of the $YFCC100M$ dataset we have used 5000 images from the CelebFaces Attributes (CelebA) Dataset from Kaggle. This dataset consists images of celebrity faces as inputs and a 10-feature output dataframe which has 10 face markers used to map the facial attributes of the person such as the x, y coordinates of both eyes, ears, nose and mouth ends. After preprocessing steps, such as normalization and rescaling, we have use transfer learning method. We called the MobileNet model Keras API and added a global average pooling layer and a relu-based dense output layer with 10 neurons i.e. our 10 face attribute markers. On training, we obtain a mean absolute percentage error of 4.2%, i.e. accuracy of nearly 96%. The results obtained are as shown in Table IV and Fig 3. We also showcase a sample model prediction in Fig 3.

Training time per epoch on average = 35 seconds.

Total training time over all model iterations = 2 hours.

4) Object Detection: In this experiment, we train the MobileNet on the COCO dataset for object detection. Due to computational constraints, we have used 5000 images from the COCO dataset. For the input,we had a variety of images where each image can have multiple distinct objects. The output can be defined as a 4-feature output dataframe where each row

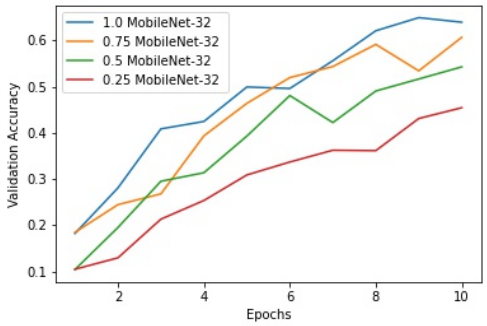

Fig. 1. Effect of Changing Width Multiplier α

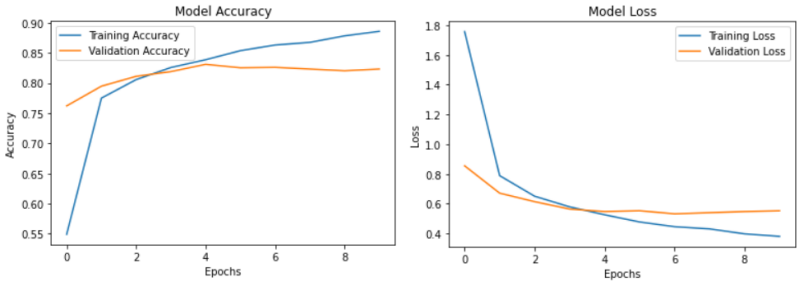

Fig. 2. Fine-Grained Recognition

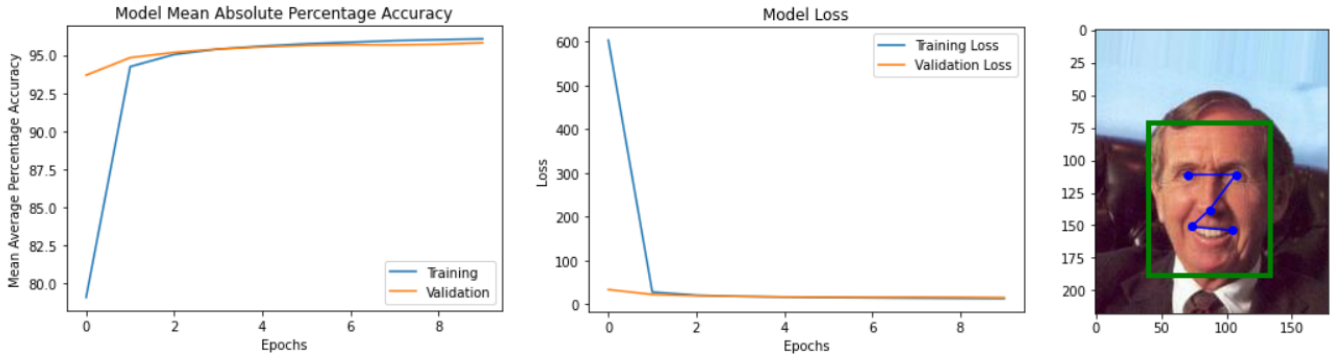

Fig. 3. Face Attributes

TABLE IV
FACE ATTRIBUTE CLASSIFICATION USING THE MOBILENET

MODEL	Mean AP (Trained on 5000 images from CelebA Dataset)	Mean AP from Paper (Trained on YFCC100M Dataset)
1.0 MobileNet-224	96%	88.7%
0.5 MobileNet-224	94.2%	88.1%
0.25 MobileNet-224	93.3%	87.2%
1.0 MobileNet-128	91%	88.1%
0.5 MobileNet-128	90.7%	87.7%

TABLE V
OBJECT DETECTION USING THE MOBILENET

MODEL	Root Mean Square Error (Trained on 5000 images from COCO dataset)	Avg. mAP Over different frameworks from Paper (Trained on entire COCCODataset)
MobileNet	122.82	18%

represents the ($xmin$, $xmax$, $ymin$, $ymax$) for an object in each image, i.e., bounding box coordinates for each object. We use transfer learning and called the MobileNet model usng Keras API and added a global average pooling layer and a Relu-based dense output layer with 4neurons i.e. our 4 bounding box coordinates. We showcase root mean squared error (RMSE) of 122.82 after training of our model. The results have been showcased in Table V and Fig 4.

5) Face Embeddings: In this experiment, we implement knowledge distillation, i.e., process of transferring knowledge from a large model to a smaller one. To implement this, we use MobileNet as our teacher architecture and a simple 5-layer CNN as the student architecture upon which MobileNet will distill its learning. We use the MNIST dataset for training. We train the MobileNet model on the dataset followed by distillation or transfer of the learning by the teacher architecture upon the simpler student CNN model. Using this method, we can see our student architecture yields an accuracy of 96.4%. To check the efficacy of this method we also train the student CNN architecture directly on the dataset and we obtain an accuracy of 97.67%. Since the results from both methods are comparable, we can conclude knowledge distillation is highly effective. The results obtained are as shown in Fig 5 which displays a comparison of results obtained by distillation learning (from teacher MobileNet architecture to student CNN architecture) vs training the student CNN architecture directly.

B. Performance analysis and insights gained

In this section, we analyze the performance of our experimentation's and provide insights regarding the same. The training of the MobileNet architecture with different values of width multiplier and resolution multiplier confirms the

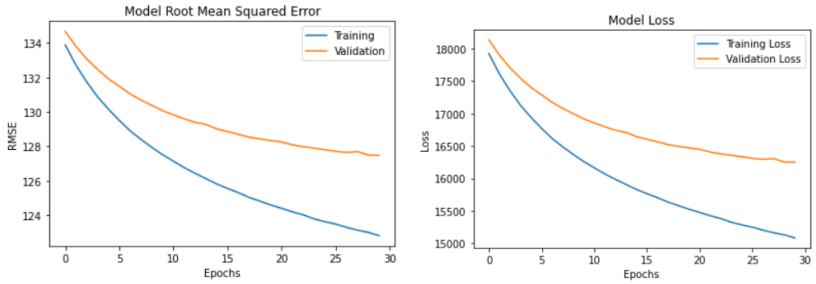

Fig. 4. Object Detection

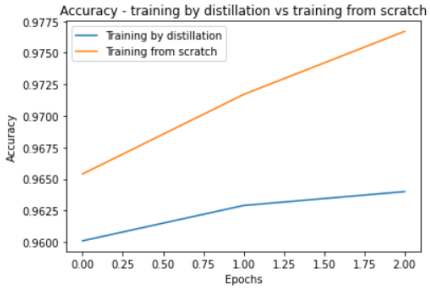

Fig. 5. Face Embeddings

relationship between the accuracy of the model and the stated set of two hyperparameters, such that the reduction in values of either alpha or resolution hyperparameter leads to model accuracy reduction. Thus, it can be concluded that regularizing these hyperparameters is effective for our architecture.

Further, we analyze the effect of incorporating depthwise separable convolutions in our architecture. We observe a significant reduction in the training time of MobileNets in comparison to architecture with standard convolutions without evident loss of accuracy from Fine-Grained Recognition and Face Attribute experiments.

We also infer from experiment 5 that knowledge distillation is a highly effective way of transferring pattern learnings from a large model to a smaller model so as to utilize the capacity of the model optimally. Thus, we can also conclude that MobileNets prove to be a good choice to be set up as a teacher architecture to transfer learnings to smaller networks as they can be trained upon complex datasets efficiently.

C. Technical challenges faced

In this section, we discuss a few major technical issues and problems faced by us in conducting the experimentation for this project. Training and processing of large datasets require adequate computational resources in order for streamlined functionality which was lacking in our case but we still managed to make the best use of the resources available to us and provide significant results. As mentioned, due to a lack of sufficient resources, we encountered several out-of-memory or RAM overload issues when trying to train the model, especially on large datasets such as the COCO dataset. Dealing with these issues was imperative to successfully completing this project, especially in a reasonable timeframe for which we had to heavily subsample from the original datasets and perform experimentation with only a limited range of hyperparameters. As a result of which we had to train the model for less than 30 epochs with lesser hyperparameters which was a major limiting factor but we still managed to showcase good results of our work.

IV. Conclusion

In this project, we studied and implemented the MobileNet architecture showcasing its efficiency and simplicity with extensive experimentation based on Transfer Learning with ImageNet weights and the performance on different application-based tasks. The experimentation results conclude that the experiments conducted in this project have a significant and comparable performance to that of the original paper [7] incorporating similar parameters and working on the same datasets to showcase and prove the model's efficiency and low latency. In the future, we aim to work on implementing MobileNets for various other mobile and vision applications like Large Scale Geolocalization, etc. We would also like to experiment with customizing the convolutional layers in addition to depthwise separable convolutions to further reduce the computations and model size without trading off accuracy.

V. Individual Student Contributions in Fractions

TABLE VI
Student Contributions

	ac5166	yb2540	aap2239
Last Name	Chitrey	Bansal	Patange
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Implemented Fine-Grained Recognition and Face Attributes Experiments	Implemented Object Detection and Face Embeddings experiments	Implemented Model Shrinking Parameters experiment
What I did 2	Documentation	Documentation	Documentation

References

[1] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 2818–2826.

[2] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," CoRR, vol. abs/1602.07261, 2016. [Online]. Available: http://arxiv.org/abs/1602.07261

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," CoRR, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[4] J. Jin, A. Dundar, and E. Culurciello, "Flattened convolutional neural networks for feedforward acceleration," 2014. [Online]. Available: https://arxiv.org/abs/1412.5474

[5] M. Wang, B. Liu, and H. Foroosh, "Factorized convolutional neural networks," in 2017 IEEE International Conference on Computer Vision Workshops (ICCVW), 2017, pp. 545–553.

[6] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," CoRR, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," CoRR, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[8] L. Sifre and S. Mallat, "Rigid-motion scattering for texture classification," CoRR, vol. abs/1403.1687, 2014. [Online]. Available: http://arxiv.org/abs/1403.1687

[9] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," online: http://www.cs.toronto.edu/kriz/cifar.html, vol. 55, no. 5, 2014.