CNN for CIFAR-10 dataset

João Mourão - 102578
DETI
Universidade de Aveiro
jmourao@ua.pt

Pedro Rodrigues - 92338
DETI
Universidade de Aveiro
ped.rodr@ua.pt

***Abstract*—Convolutional Neural Networks (CNNs) have revolutionized image classification tasks, and the CIFAR-10 dataset stands as a standard benchmark for evaluating these models. In this study, we delve into the efficacy of CNN architectures specifically tailored for the CIFAR-10 dataset. Through extensive experimentation and analysis, we explore the design and performance of various CNN models, optimizing their configurations to achieve superior accuracy in classifying the diverse 32x32 color images across ten distinct classes. Leveraging techniques such as data augmentation, residual networks, and architectural modifications, we aim to uncover the most effective strategies for enhancing CNN performance on the CIFAR-10 dataset.**

***Index Terms*—CIFAR-10, CNN, Machine Learning, Deep Learning**

I. Introduction

Convolutional Neural Networks (CNNs) have redefined image classification, using deep learning to decode complex visual data. The CIFAR-10 dataset, which contains diverse, low-resolution images from ten categories, serves as a standard for evaluating CNNs' capacity to interpret detailed visual information.

This paper explores CNNs applied to CIFAR-10. We investigate CNN architectures, their feature extraction capabilities, and strategies for improving classification accuracy. By analyzing CNN performance on this dataset, we hope to understand their adaptability and limitations in real-world image classification.

II. State of the art

The CIFAR-10 dataset was introduced by the Canadian Institute for Advanced Research (CIFAR) as a benchmark dataset for the machine learning and computer vision communities. The exact year of its release was in 2009. It was made publicly available as a standard dataset for research purposes to facilitate the development and evaluation of machine learning algorithms, particularly in the field of image classification and object recognition .

By searching up what were the models with the best results, we were able to find many papers with great results with various approaches to the problem. One of these papers was titled Deeply-Supervised Nets (DSN) [4]. DSN brought transparency to CNNs by showing the role of intermediate layers in classification, and consequently having a clearer understanding of feature representation. In addition, DSN introduced "companion objectives" to hidden layers, diverging from traditional pre-training methods, guiding each layer explicitly towards the classification goal. DSN showcased substantial performance enhancements over existing techniques at the time on benchmark datasets such as CIFAR-10, reaching an accuracy of 91,8%.

In another paper [5], we learned about the concept of Fractional Max-Pooling that revolutionized spatial pooling in convolutional neural networks. Departing from the conventional alpha times alpha max-pooling with alpha set to 2, Fractional Max-Pooling allowed non-integer values for alpha, enabling diverse pooling regions and mitigating rapid size reduction problems seen in the previously known approaches. This method significantly reduced overfitting across datasets, beating the previous accuracy with a value of 96.5%.

The following paper introduced GPipe[6], a pipeline parallelism library designed to efficiently scale up neural network capacity beyond the memory limits of a single accelerator. GPipe enables the scaling of diverse network architectures by leveraging pipeline parallelism, dividing layers across multiple accelerators. This approach offers task-independent model parallelism, unlike architecture-specific solutions. Utilizing a novel batch-splitting pipelining algorithm, GPipe achieves nearly linear speedup when partitioning models across accelerators. The success of GPipe was proven when it was combined with Transfer Learning and obtained 99% accuracy.

The next paper [7] introduced BiT, or Big Transfer, a new method for training neural networks to understand images. They found that by first teaching the network on big collections of labeled pictures and then fine-tuning it for specific tasks, they could make it obtain really good results. BiT performed amazingly well across various datasets, even when there were super few examples to learn from. It achieved 87.5% accuracy on one of the big image datasets (ILSVRC-2012), 99.4% on CIFAR-10, and 76.3% on a set of 19 different tasks. Even on smaller datasets with just 10 examples per class, BiT managed 76.8% accuracy on ILSVRC-2012 and a whopping 97.0% on CIFAR-10.

The current holder of the best accuracy for CIFAR-10 [8], is a paper where we learned about Vision Transformers(or ViT). ViT achieves impressive performance on image classification tasks when pre-trained on extensive data and transferred to various mid-sized or small image recognition benchmarks like ImageNet, CIFAR-100, and VTAB. ViT outperformed the previous state-of-the-art CNNs while demanding significantly fewer computational resources for training. This showcases the viability of transformers in computer vision tasks and

highlights their potential as a strong alternative to CNN-based approaches. In the end, it was able to reach 99.5% accuracy on CIFAR-10.

III. Data Visualization

From the beggining we know that CIFAR-10 consists in 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50.000 training images and 10.000 test images. In this section we are going to visualize that information.

In the figure 1 we can see that after loading the data set and checking the data obtained we can see that we have the previsous said 50000 training images and 10.000 test images. We can also see that we have 10 different classes, those being, 'airplane', 'automobile' ,'bird' ,'cat' ,'deer' ,'dog' ,'frog' , 'horse', 'ship', 'truck'.

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train, X_test = X_train/255, X_test/255 # Normalizar os dados

classes = ['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck']

y_train =  tf.keras.utils.to_categorical(y_train)
y_test =  tf.keras.utils.to_categorical(y_test)
num_classes =  y_test.shape[1]


print("Number of classes: "+str(num_classes))
print("Number of training classes: "+str(len(X_train)))
print("Number of test classes: "+str(len(X_test)))
```

```
Number of classes: 10
Number of training classes: 50000
Number of test classes: 10000
```

Fig. 1. Visualization of Training Images, Test Images and Number of classes

Following that, all that was left to see was the images itsel. For that we only listed 3 of them with the name of the corresponding class above it

IV. Data Preprocessing

First of all we had to load the data set, for that we developed a function called "load dataset" We can see here that we load the CIFAR-10 data set to the variables 'trainX', 'trainY', 'testX' and 'testY'. After that we one-hot enconded the target labels 'trainY' and 'testY', one-hot enconding is a process of converting categorial labels, in this case class labels, into a binary matrix where each row corresponds to a sample and each column corresponds to a class. In the end we return the data.

Following that we had to normalize the data, for that we also created a new function, called "normalize_data". Here we can see that the input data are being converted to data type "float32". This is a very good practice because we it's oftern beneficial to work with floating-point numbers when performing normalization. The "normalization" is done

```
In [19]: fig = plt.figure(0)
         fig.set_size_inches(10, 10)

         for i in range(3):
             fig.add_subplot(330+1+i)
             plt.imshow(X_train[i])
             plt.title("{}".format(classes[i]))

         plt.show()
```

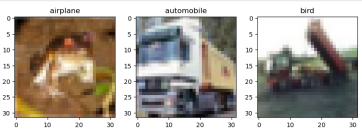

Fig. 2. Visualizing images

```
# load train and test dataset
def load_dataset():
  # load dataset
  (trainX, trainY), (testX, testY) = cifar10.load_data()
  # one hot encode target values
  trainY = to_categorical(trainY)
  testY = to_categorical(testY)
  return trainX, trainY, testX, testY
```

Fig. 3. Function Load Data Set

```
# normalize data
def normalize_data(train, test):
  train_norm = train.astype('float32')
  test_norm = test.astype('float32')
  train_norm = train_norm / 255.0
  test_norm = test_norm / 255.0
  return train_norm, test_norm
```

Fig. 4. Function to Normalize data

dividing the train_norm and the test_norm by 255. This scales the pixel values to be between [0,1]. In the end we return the values.

Just like we did in the previous project to get better results we used a technique called "data augmentation" to artificially increase the size of the training dataset by applying various transformations. The transformations applied here were, randomly shift the width of the images by 10%, randomly shift the height of the images by 10% and randomly flip images horizontally.

```
#data augmentation
datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
it_train = datagen.flow(trainX, trainY, batch_size=32)
steps = int(trainX.shape[0] / 64)
```

Fig. 5. Data Augmentation

Also here, the variable "steps" is calculated to determine the number of batches per epochs.

V. Models

A. CNN

Before moving on to how Convolutional Neural Networks are structured, a short explanation of the layers used is necessary. Along this project we used Input Layers, that are responsible for receiving the data, we used ZeroPadding layers, that are responsible for adding rows and collumns of zeros around the data that we want to process in order to preserve the spatial information of the input during convolutional operations, we used Convolutinal Layers, that convolves and processes data with a set of filters, we used Max Pooling layers, that are responsible for reducing the spatial dimensions of the feature maps, extracting important features, and improving the computational efficiency of CNNs, we used Dropout layers, that are a regularization technique that helps prevent overfitting by randomly setting to zero a portion of the neurons in the previous layer during training, we used Flatten layers, that reshapes the multidimensional feature maps into a one-dimensional vector preserving the depth, we used

Dense layers, that are fully connected layers meaning that it connects every neuron in the current layer to every neuron in the subsequent layer and we used Batch Normalization layers, that is another technique of regularization that normalizes the inputs of a neural network by adjusting and scaling the activations of each mini-batch during training.

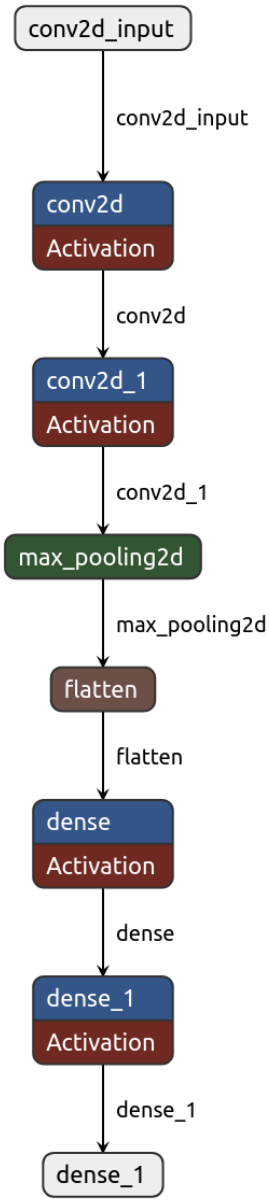

Fig. 6. Graphical representation of the first model trained

In the figure 6 we can see the first model that we trained. It's a very simple model. After running this model and analysing the graph we can see that this model rapidly overfits the test dataset. This model obtained 64.9% with 20 epochs.

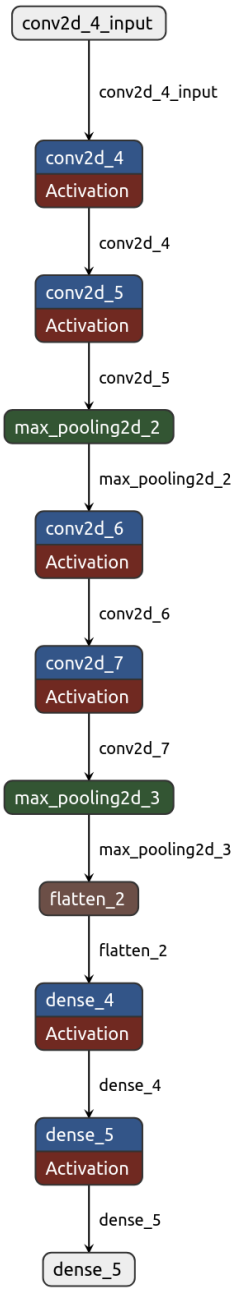

Fig. 7. Graphical representation of the second model trained

Following that we tested the model on figure 7 , we can see that adding two more Conv2d and one more Maxpool performs better then the previous model. But we keep seeing a strong overfitting. With the changes, the model reached 67.7% with 20 epochs.

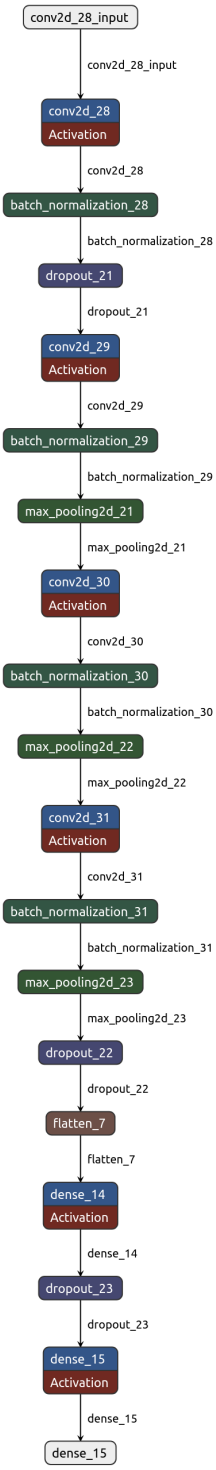

Fig. 8. Graphical representation of the third model trained

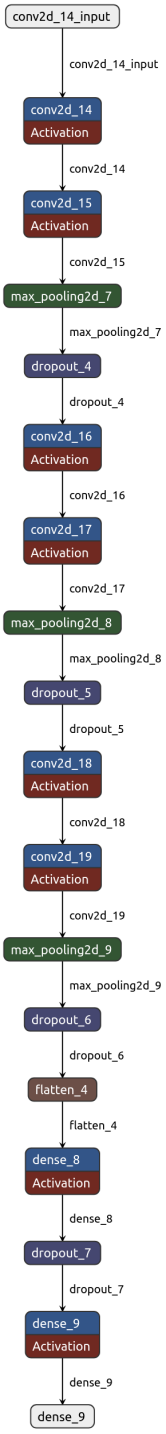

Fig. 9. Graphical representation of the fourth model trained

In the model of figure 8 we added ”batch normalization”, this is a technique designed to automatically standardize the inputs to a layer in a deep learning neural network. After these changes the accuracy increased up to 72.4% with 20 epochs. Training this model we got some improvements, so we tried stucking to this model for the rest of the trained CNN's.

After trying the model of figure 8 we decided to give one more go to the previous model (figure 7) but adding one more layer ”Dropout” after the layer MaxPool2D. With this model we obtained our best results for 20 epochs:78.1%. Since it was our best result we decided doing 100 epochs for this model and obtained 84.5%.

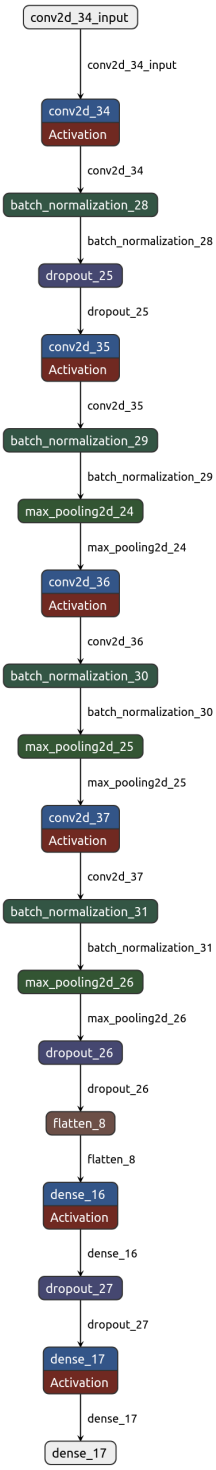

Fig. 10. Graphical representation of the fifth model trained

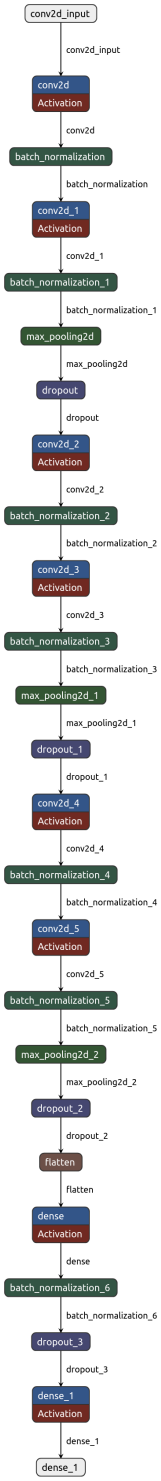

Fig. 11. Graphical representation of the seventh model trained

After that the models of the figures 10, **??** and 11 are just slight variations in dropout and batch size of the model in 8 trying to get the best CNN that we could possible can. Out of all CNN models, the best was 11 with 87.5% for 100 epochs.

B. Residual Networks

Inspired by some of the state of the art papers, we decided to give a try implementing Residual Networks (ResNet).

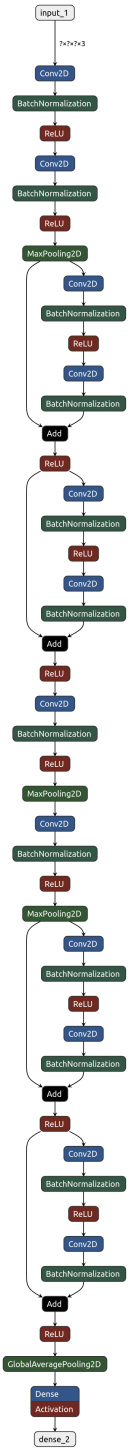

Fig. 12. Graphical representation of the best ResNet model trained

After some tries this was our best model, obtaining 91.1% accuracy after 50 epochs.

C. DLA

This algorithm was inspired in a Github Repository that was found while we were searching for state of the art information, which happened to have a high accuracy.

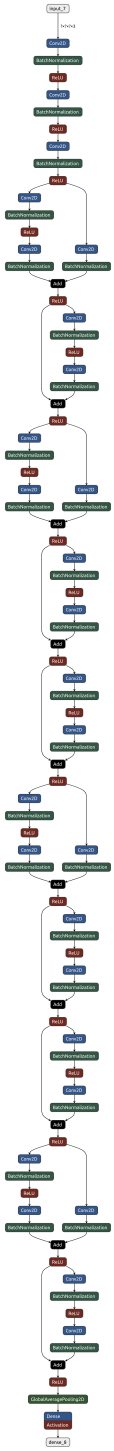

Fig. 13. Graphical representation of the best DLA trained model

The trained model ended up with 91.0% accuracy after 50 epochs.

D. Ensemble Model

Since we had three different models with considerably high accuracies, we decided that joining the models would be a good option to obatin better results. We chose soft voting as a way to elect the best model. In soft voting models provide probabilities for each class, and the average probability for each class is calculated across all models. The class with the highest average probability is chosen as the final prediction. It resulted in a model with 92.36%. Below in

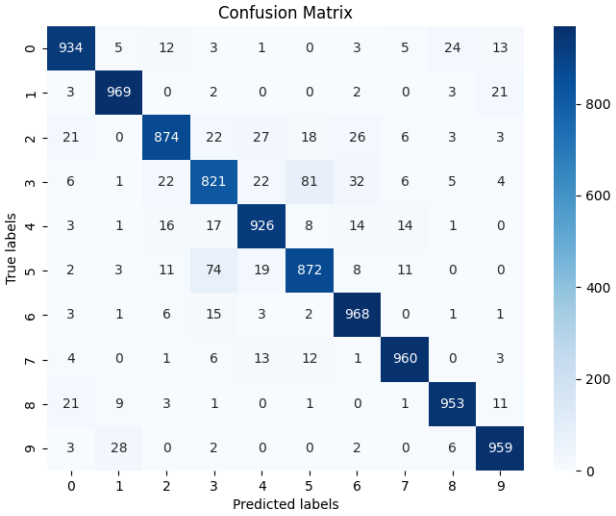

Fig. 14. Confusion Matrix for the ensembled model

RESULTS

Below, we will present the graphs for cross-entropy loss and accuracy along the epochs for the best models for CNNs,ResNets and DLAs and a table with all the highest accuracies including the state of the art.

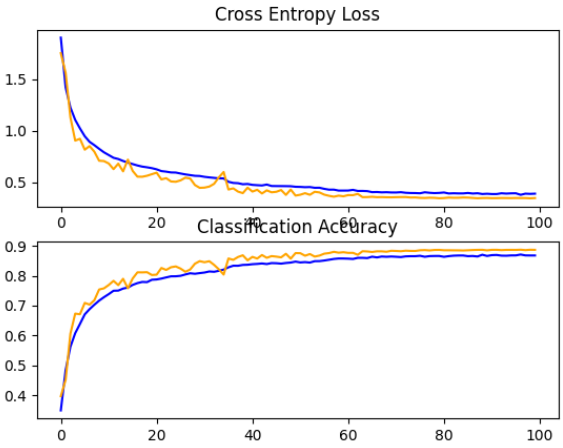

Fig. 15. Cross-entropy loss and accuracy of the best CNN trained model

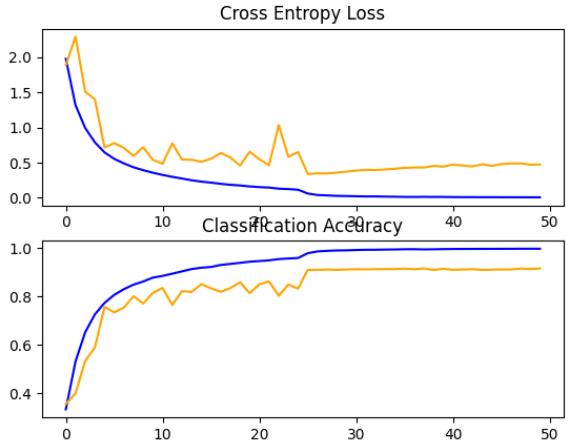

Fig. 16. Cross-entropy loss and accuracy of the best ResNet trained model

Model	Test Accuracy
CNN	87.47%
ResNet	91.12%
DLA	91.01%
Ensembled Model	92.36%
Current State of the art	99.50%

TABLE I
ACCURACY FOR THE DIFFERENT MODELS

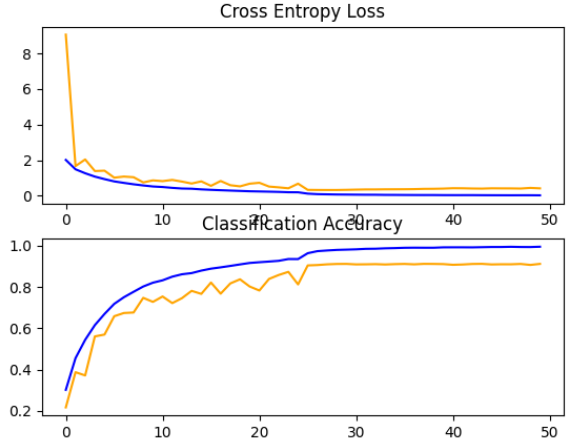

Fig. 17. Cross-entropy loss and accuracy of the best DLA trained model

Conclusion

[h] Comparing our results to the state of the art results, we are still some accuracy points below. By using Visual Tranformers we could have perhaps shortened the distance between these two models. Despite this, the project was still benefitial to us since we learned about algorithms we had never explored before.

Work Division

References

[1] Ektasharma. *Simple CIFAR-10 CNN Keras Code with 88% Accuracy*. Kaggle. URL: https://www.kaggle.com/code/ektasharma/simple-cifar10-cnn-keras-code-with-88-accuracy

[2] Jason Brownlee. *How to Develop a CNN from Scratch for CIFAR-10 Photo Classification*. Machine Learning Mastery. URL: https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/

[3] Kuang Liu. *PyTorch CIFAR Repository - DLA Model*. GitHub. URL: https://github.com/kuangliu/pytorch-cifar/blob/master/models/dla.py

[4] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, Zhuowen Tu. *Deeply-Supervised Nets*. 2014. arXiv eprint: https://arxiv.org/abs/1409.5185. *arXiv:1409.5185 [stat.ML]*.

[5] Benjamin Graham. *Fractional Max-Pooling*. 2015. arXiv eprint: https://arxiv.org/abs/1412.6071. *arXiv:1412.6071 [cs.CV]*.

[6] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, Zhifeng Chen. *GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism*. 2019. arXiv eprint: https://arxiv.org/abs/1811.06965. *arXiv:1811.06965 [cs.CV]*.

[7] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, Neil Houlsby. *Big Transfer (BiT): General Visual Representation Learning*. 2020. arXiv eprint: https://arxiv.org/abs/1912.11370. *arXiv:1912.11370 [cs.CV]*.

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv eprint: https://arxiv.org/abs/2010.11929. *arXiv:2010.11929 [cs.CV]*.