Fashion MNIST classification using PyTorch Convolution Neural Network with Cross-Validation

Fatemeh Sharifi
Department of Mechanical Engineering
Boston University
fatima94@bu.edu

Nanna Katrin Hannesdottir
Department of General/Computer Engineering
Boston University
nannkat@bu.edu

Shahaf Dan
Department of Computer Science
Boston University
shahaf@bu.edu

Abstract

The following paper discusses the application of Convolutional Neural Networks [CNN] in machine learning and image recognition, as well as result analysis and outcome optimization using cross-validation. The spoken methods will be define and explained in their application to the Fashion-MNIST dataset. The main question and thesis around the paper will revolve around asking whether or not CNN model construction and cross-validation analysis can increase accuracy rates and performance success. The paper will also discuss the results, as well as present them in various visualisation graphics, including a confusion matrix, cross-validation increase rate representation, and image processing performance.

1 Introduction

1.1 Essentials and Foundations

The Fashion-MNIST is an Image dataset for benchmarking Machine Learning Algorithms. It provides it's user with 60,000 training samples and 10,000 testing examples, and is often used as an introduction construction dataset to model application on more complex apparel-related data sets. Prior to our progamming work, we have established some initial and essential research about the topics our paper discusses (and their application to the universally used and acceptable Kaggler dataset: Fashion-MNIST. The article (1) discusses and proposes three different convolutional neural network architectures and model designs, and uses batch normalization and residual skip connections for the acceleration of the learning-optimization process. Results on the Fashion-MNIST dataset include improved accuracy of around 2% and similar decrease in loss.

The paper (2) explores Hierarchical Convolutional Neural Network (H-CNN) on the Fashion-MNIST dataset for benchmarking records. Among the conclusion, is that H-CNN brings better performance in classifying clothing items (decreased and loss, and greater accuracy), which we will attempt in applying through our program. The scholar article (3) ellaborates on the topic of model trainig and image recosntruction from the pose parameters and identity of the correct top-level capsule. In it's summary there lies that conclusion that setting a threshold on the L_2 norm distance between the input image and its reconstruction from the winning capsule is very effective at detecting adversarial images, which also works for convoltuional neural networks that have been trained to reconstruct the image from various different layers.

As part of our final project for the Summer 2021 CS542 [Machine Learning] class at Boston University, we have decided collectively to apply our knowledge into Convolutional Neural Networks [CNN] model construction with image recognition. Our goal for the project was to design a program to recognize clothing items from an additional dataset and label them accordingly, using the training and testing samples from the Fashion-MNIST dataset. The Fashion-MNIST dataset is made of 60,000 training samples, where each tuple (row, object, item of the dataset), is constructed of a 785 attributes: 1 label with the according and matching clothing items from the 10 unique labels in the dataset, and 784 additional attributes marking 784 pixels, which through PyTorch (Machine Learning framework for python), is converted accordingly into a 4-D tensor object as a 1000 x 1x 28 x 28 pixels image.

Using our algorithm, we hoped to achieve, and managed to achieve, the construction of a CNN model, trained and tested by the Fashion-MNIST dataset. Our model successfully recognized tested for a range of 87% - 94% item recognition and classification by applying it to an additional dataset of apparel items worn on people (multilayer images). Behind our algorithm, our approach to program the CNN model construction and testing is explained in the following section. In the section 2,we deefine our Convolutional Neural Network Model [CNN] using a python machine learning framework [PyTorch] that is run on a SCC GPU.Then, How we used Cross-Validation to save the best model trained is investigated in the third section.Last section is assigned to visualize our results using a confusion matrix, loss and accuracy graphic presentations.

2 Developing the Convolutional Neural Network

2.1 Choosing CNN

The model built and used in this project was a custom-designed Convolutional Neural Network or CNN. Convolutional Neural Networks have been used extensively in the field of image recognition throughout the years.

CNNs are designed with the aim of picking up patterns in the input images irrespective of subtle or drastic transformations of the content. A key requirement for Machine Learning algorithms in image recognition is to develop a model that ensure invariance of the model predictions despite transformations in the data. It is evident that the identity of an image input does not depend on rotation or skew, the image can be taken from different angles and in different contexts. Ideally, significant changes in the raw input data should therefore not change the output of the classification system.

Invariance can be achieved through different approaches, which can broadly be defined into four categories: 1. Augmentation of the training set using transformed replicas 2. Introducing a regularization term that penalizes changes in output 3. Extracting invariant features in the pre-processing phase and 4. Building the invariance into the structure of the neural network itself (4).

Convolutional Neural Networks belong to the last category and incorporates invariance by extracting and compressing local features through usage of kernel filters and subsampling. The algorithm builds on the fact that nearby pixels are more strongly correlated than ones further away and the intuition that local features prominent in one feature are likely to be prominent elsewhere (4).

2.2 Model structure

With CNN being widely popular, various tools such as PyTorch and Tensorflow offer both tools and pretrained models to easily develop CNN algorithms. Since this project focuses on the Fashion MNIST a dataset with fairly atypical images: 28x28x1, it did not seem apropriate to use a pretrained model, as most of those are trained on much more complex datasets. Therefore a decision was made to develop a custom Convolutional Neural Network using building blocks from the `Torch.nn` library, documentation found at

`https://pytorch.org/docs/stable/nn.html`

In line with the convention for CNN, our model is composed of two main phases, a feature extraction phase with convolutional layers and pooling layers and then a classification phase, which is composed of fully-connected linear layers, with a single dropout layer to prevent overfitting.

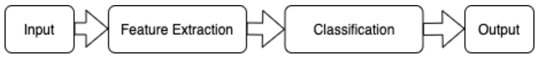


Figure 1: Flow of the model

2.2.1 Convolutional layers and Pooling

The feature extraction of a CNN is composed of pairs of convolutional layers and subsampling layers. The convolutional layers create feature maps of the input image using one or more feature kernels that are applied to sub regions of the image. The subsequent subsampling speeds up computation in the network and helps with invariance as smaller samples are less sensitive to variants in the input (4).

The function `Torch.nn.Conv2d()` from PyTorch is a compact tool to create such a layer. Taking as arguments the *number of input channels* (1 if grayscale, 3 if RBG) *number of output channels*(i.e. dimension of feature maps), and *kernel size* (along with other optional keyword arguments) it computes feature maps from the given conditions. In the simplest case with input layer (N, C_in, H, W) and output (N, C_out, H_out, W_out) the process can be described with the equation

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) \star input(N_i, k) \tag{1}$$

Where $\star$ is the cross-correlation operator, N is the input size (batch size), C is the number of channels, H and W are height and width and k denotes the kernel/filter.

The output of the feature maps serve as inputs to subsampling or 'pooling' layers. The pooling process samples subregions of the feature map and translates their contents to a smaller region in the pooling layer. This is for instance done by taking input from a 2x2 region of the image and keeping only the maximum pixel value of that region. In other words max-pooling Our choice for pooling was PyTorch `MaxPool2d()` which takes *kernel size* and *stride* as argument along with other optional keyword arguments. The default value for *stride* is 2, meaning the image size is reduced by half.

In our custom CNN we included two sets of convolutional layers and corresponding pooling layers:

Convolutional Layer 1

- `Conv2d()`, 1 input channel, 32 output channels, kernel size of 3 and a padding of 1.
- `MaxPool2d()`, kernel size 2 and a stride of 2

Convolutional Layer 2

- `Conv2d()`, 32 input channels, 64 output channels, kernel size 3 and 0 padding.
- `MaxPool2d()`, kernel size 2 and a stride of 2

By increasing the dimension of the output channels, first to 32 and then to 64, the idea was to train the model to pick up on different types of features, as for each output channel a different kernel is applied.

2.2.2 Regularization of Feature Layers

We wrapped the output of the `Conv2d()` layers in both a nonlinear ReLU activation function and a normalization function.

The danger of over-fitting is present in convolutional layers just as in linear layers.The well known ReLU function outputs 0 if the input is negative and passes the input through otherwise, $f(x) = x^+ = max(0, x)$. We used it's PyTorch correspondant `Torch.nn.ReLU()`, to decrease linearity in our feature extraction.

By using PyTorch's `Torch.nn.BatchNorm2d()` one can create normaliztion on a batch-by-batch basis, where mean and standard deviation are calculated per dimension. The function used is $y = \frac{x-\mathbf{E}[x]}{\sqrt{\mathbf{V}[x]+\epsilon}} * \gamma + \beta$ Where γ and β are themselves learnable parameter vectors of the same length as the input.Normalizing the batches increases standardizaion of the data and increases the efficiency of the training process (5).

2.2.3 Linear layers

The final step is the classification process, typically done with a series of fully-connect linear layers, each layer containing M combinations of the input variables $x_1, x_2, ...x_D$ in the form

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{2}$$

Where $j = 1, ...M$ and the superscript (1) denotes the layer number (in this case first layer) (4).

In our implementation we included 3 of those simple linear layers along with a so called dropout layer for regularization, better discussed in the result section.

It is worth noting that to get the output, a single "softmax" layer is typically included after the linear layers as the last step of the classification to get the probabilities of each class. Softmax is a generalization of the logistic sigmoid function that takes in a vector of real number and normalizes it: $softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))}$. For us, this layer was embedded in our loss function class, PyTorch's `nn.CrossEntropyLoss()`.

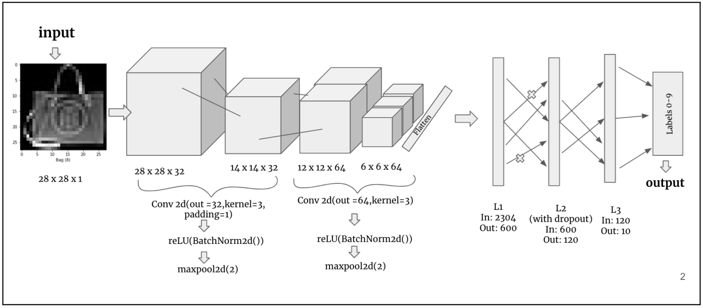


Figure 2: Convolutional Neural Network for classifying Fashion MNIST

3 Cross Validation

In the process of validation, we need a separate data set for testing the model during the training to find out which parameter has the best performance. However, we didn't have a separate data set for validation in this data set. Because of this reason, we used cross-validation which is a re-sampling procedure used to evaluate machine learning models on a limited data sample. For this project, we used k-fold cross-validation.

The only parameter in this procedure is k which define the number of group that we are gonna split the training data. K-fold cross-validation is a popular method because it is simple to understand and because it generally results in a less biased estimate of the model skill than other methods, such as a simple train/test split. We generally can explain this method as Algorithm 1.

Algorithm 1: k-fold cross validation

```
1. Shuffle the train data set randomly.
2. Split the train data set into k groups.
3. For each unique group:
     • Take the group as a validation data set
     • Take the remaining groups as a training data set
     • Fit a model on the training set
     • Evaluate it on the validation set
     • Retain the evaluation score
4. Determine the parameters of the model using best model evaluation scores
```

3.1 The effect of parameter k

The k value must be chosen carefully for your data sample. A poorly chosen value for k may result in a misrepresentative idea of the skill of the model, such as a score with a high variance (that may change a lot based on the data used to fit the model), or a high bias, (such as an overestimate of the skill of the model).The value for k is chosen such that each train/test group of data samples is large enough to be statistically representative of the broader dataset.

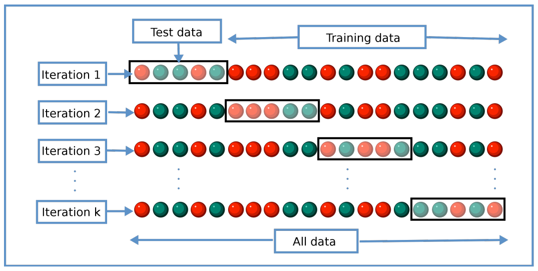


Figure 3: k-fold cross validation

The choice of k is usually 5 or 10, but there is no formal rule. As k gets larger, the difference in size between the training set and the resampling subsets gets smaller. As this difference decreases, the bias of the technique becomes smaller(6). To summarize, there is a bias-variance trade-off associated with the choice of k in k-fold cross-validation. Typically, given these considerations, one performs k-fold cross-validation using $k = 5$ or $k = 10$, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance(7).

4 Result

Because of the size of the training data set and the short time we had for the project, we choose the size of batch to be 1000 data and the k parameter of the cross-validation to be 3. By these choices, we could run the code faster and debug them easier.

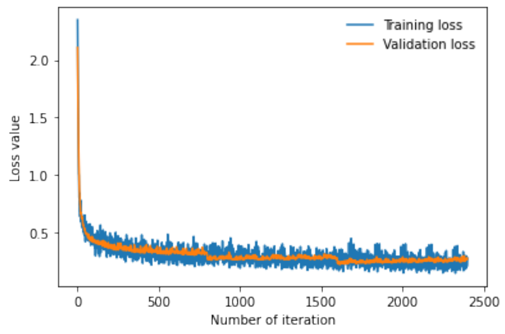
4.1 prevent over-fitting

One of challenges in this project was to prevent the over-fitting in our model that we try to overcome it by using $L2$ regularization and dropout regularization. The weight decay parameter for $L2$ regularization is determined as 10^{-5} in the optimization cost that make the solution smoother and prevent over-fitting in the model. Moreover, we used Adam optimizer which works well in practice and compares favorably to other stochastic optimization methods (8).

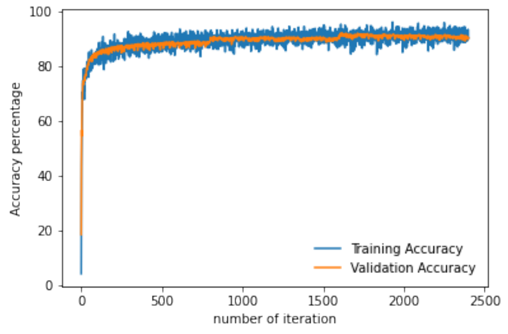
In addition, we uses Dropout regularization which is a regularization method that approximates training a large number of neural networks with different architectures in parallel. During training, some number of layer outputs are randomly dropped out. This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different view of the configured layer (9).We choose the dropout ratio to be 0.25 in this project.

4.2 Accuracy and Loss of the model

As you can see in the figure 4a, the training and validation error or loss is close to each other which means we don't have over-fitting in our model. We expect that if we could run the algorithm for more iteration, we would get a less error at the end. Also for accuracy of our model, we show that in the figure 4b we reached the 88 percentage accuracy at the end of the training procedure.



(a) Loss value of training and validation



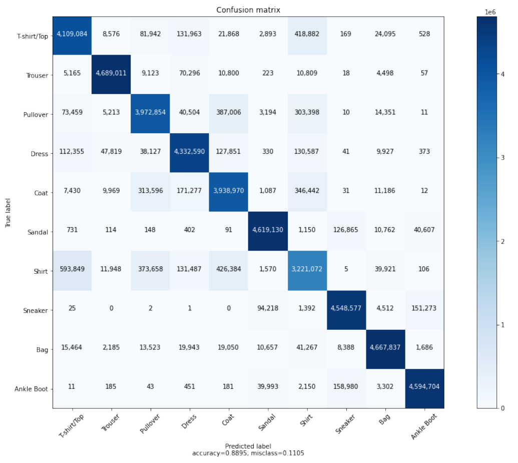
(b) Accuracy percentage of training and validation

Figure 4: The result of training process

4.3 Confusion Matrix

For visualizing the accuracy of our model, we showed the confusion matrix for validation data and testing data in the figure 5a and 5b. As you can see, the main diagonal in both of the confusion matrix have a much greater value rather than other elements which represent the accuracy of the model.

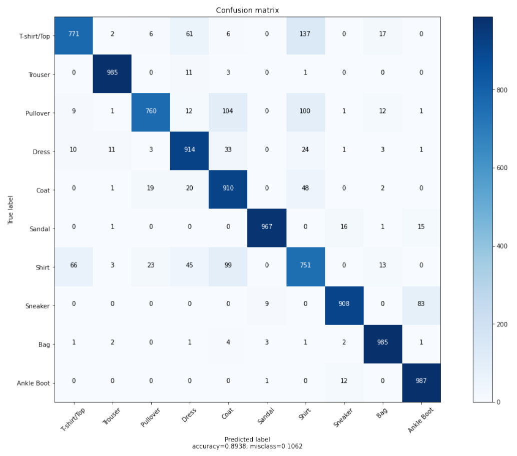
In the both cases, the minimum accuracy is related to the class of shirt, T-shirt, and pullover that have a kinda of similar shape. In the other hand, we have the maximum accuracy in the class of Trouser and bag in the both case of validation and testing data sets.


Confusion matrix

True label	T-shirt/Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
T-shirt/Top	4,109,084	8,576	81,942	131,963	21,868	2,893	418,882	169	24,095	528
Trouser	5,165	4,689,011	9,123	70,296	10,800	223	10,809	18	4,498	57
Pullover	73,459	5,213	3,972,854	40,504	387,006	3,194	303,398	10	14,351	11
Dress	112,355	47,819	38,127	4,332,590	127,851	330	130,587	41	9,927	373
Coat	7,430	9,969	313,596	171,277	3,938,970	1,087	346,442	31	11,186	12
Sandal	731	114	148	402	91	4,619,130	1,150	126,865	10,762	40,607
Shirt	593,849	11,948	373,658	131,487	426,384	1,570	3,221,072	5	39,921	106
Sneaker	25	0	2	1	0	94,218	1,392	4,548,577	4,512	151,273
Bag	15,464	2,185	13,523	19,943	19,050	10,657	41,267	8,388	4,667,837	1,686
Ankle Boot	11	185	43	451	181	39,993	2,150	158,980	3,302	4,594,704

Predicted label
accuracy=0.8895; misclass=0.1105

(a) validation


Confusion matrix

True label	T-shirt/Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
T-shirt/Top	771	2	6	61	6	0	137	0	17	0
Trouser	0	985	0	11	3	0	1	0	0	0
Pullover	9	1	760	12	104	0	100	1	12	1
Dress	10	11	3	914	33	0	24	1	3	1
Coat	0	1	19	20	910	0	48	0	2	0
Sandal	0	1	0	0	0	967	0	16	1	15
Shirt	66	3	23	45	99	0	751	0	13	0
Sneaker	0	0	0	0	0	9	0	908	0	83
Bag	1	2	0	1	4	3	1	2	985	1
Ankle Boot	0	0	0	0	0	1	0	12	0	987

Predicted label
accuracy=0.8938; misclass=0.1062

(b) testing

Figure 5: The confusion matrices

4.4 Expanding the model on new data

The Fashion MNIST contains grayscale, single stance, very simplified fashion images. As a way of doing exploratory analysis of our model, we were curious to see how (badly) it might do on a dataset with more complex and realistic fashion images. The Fashion Product Images found on Kaggle is a dataset with 44000, 60x80 color images of fashion items with a corresponding .csv containting detailed labels for each image (primary category, sub-category, sub-sub-category etc..).

`https://www.kaggle.com/paramaggarwal/fashion-product-images-small`

Unlike the MNIST the pictures in Fashion Product Images are not completely standardized, many are of models posing in the clothing in different positions and multiple types of clothing. Could our model handle that? Furthermore, Downloading the smaller resolution version we set out on a mission to preprocess the images and recode the labels of the Fashion Product Images to see how our model performed.

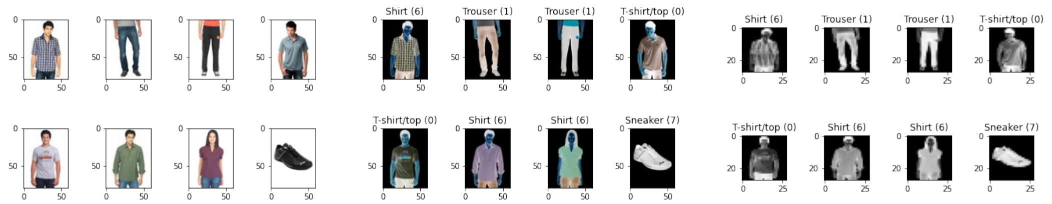


Figure 6: Image processing of the Fashion Product Images Dataset.

The convolutional neural network was designed with a 28x28 grayscale image as input and consequently some preprocessing was needed on the more complex and larger images in order for them to be valid inputs to the system. Using tools from the `pillow` libary in Python and PyTorch `torchvision` library tools we first inverted the color scheme of the images to make them match the white-on-black format of the MNIST data, we then made them grayscale and finally shrank them to 28x28. Furthermore a normalization transform was applied before the testing.

For the target labels, the labels of the new data had to be recoded to fit into the 10 categories of the MNIST data. The Fashion Product Images contained some categories that were not appliccable like 'jewelry' and 'swimsuits' which had to be filtered out. The sub-categories furthermore had to be matched with the closest category in MNIST. Resulting was an array 30175 images remaining, with the following coding scheme: T-shirt/top(0): 'Tshirts', 'Tops', Trouser(1):'Jeans', 'Track Pants', 'Trousers', 'Rain Trousers', 'Leggings', 'Jeggings', 'Shorts', 'Capris', Pullover(2): 'Sweatshirts','Sweaters' , Dress(3): 'Dresses', 'Tunics', 'Kurtas', 'Kurtis', 'Churidar', 'Salwar', 'Lehenga Choli', Coat(4) 'Rain Jacket', 'Waistcoat', 'Blazers', 'Jackets', 'Nehru Jackets', Sandal(5): 'Flip Flops', 'Sandals', 'Sports Sandals' Shirt(6): 'Shirts', Sneaker(7): 'Casual Shoes', 'Sports Shoes', 'Flats', Bag (8): 'Handbags', 'Laptop Bag', 'Backpacks', 'Clutches', 'Duffel Bag', 'Trolley Bag', 'Mobile Pouch', 'Messenger, Bag','Rucksacks' Ankle boot(9): 'Formal Shoes','Heels'

Despite our efforts, unsurprisingly, results were not good. The accuracy jumped down to 11% with the model classifying all images as 'shirt'.

5 Conclusion

After running our algorithm on various datasets, we have seen a slight improvement by applying a convolutional neural network into our image recognition and cross-validation analysis. In numbers, we have transitioned from a success rate of 87% - 91%, into a success rate of 90% - 94%. From the confusion matrices generated (from running our program both on the validation and testing datasets), we also get to see a slight yet significant improvement in our successfully identified and predicted objects, implying on improvement in performance using CNN on 4D-tensors.

Furthermore, under our conclusion also lies the analysis of cross-validation performance. We notice that as we increase the number of iterations, both our validation and training loss decrease. Similarly, we notice that as the number of iteration increase, our accuracy rates increase in average (both for validation and training). Therefore, all in all we can conclude that by using L2 and dropout regularization in our convolutional neural network model training, as we increase the number of iterations, we would achieve better results in terms of a right fit to the real model with minimum loss and maximum accuracy.

References

[1] S. Bhatnagar, D. Ghosal, and M. H. Kolekar, "Classification of fashion article images using convolutional neural networks," in *2017 Fourth International Conference on Image Information Processing (ICIIP)*, 2017, pp. 1–6.

[2] Y. Seo and K. shik Shin, “Hierarchical convolutional neural networks for fashion image classification,” *Expert Systems with Applications*, vol. 116, pp. 328–339, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417418305992

[3] N. Frosst, S. Sabour, and G. Hinton, “Darccc: Detecting adversaries by reconstruction from class conditional capsules,” *arXiv preprint arXiv:1811.06969*, 2018.

[4] C. M. Bishop, “Pattern recognition,” *Machine learning*, vol. 128, no. 9, 2006.

[5] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[6] M. Kuhn, K. Johnson *et al.*, *Applied predictive modeling*. Springer, 2013, vol. 26.

[7] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[8] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.

[9] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.