

When Neural Networks Using Different Sensors Create Similar Features

Hugues Moreau, Andréa Vassilev, Liming Chen

▶ **To cite this version:**

Hugues Moreau, Andréa Vassilev, Liming Chen. When Neural Networks Using Different Sensors Create Similar Features. EAI MobiCase 2021, Nov 2021, Online, China. hal-03412575

HAL Id: hal-03412575
https://hal.science/hal-03412575

Submitted on 3 Nov 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

When Neural Networks Using Different Sensors Create Similar Features

Hugues Moreau[1,2][0000−0002−0569−4190], Andréa Vassilev[1], and Liming Chen[2][0000−0002−3654−9498]

[1] Université Grenoble Alpes, CEA, Leti, F-38000 Grenoble, France
name.surname@cea.fr
[2] Department of Mathematics and Computer Science, Ecole Centrale de Lyon, University of Lyon, Ecully, France name.surname@ec-lyon.fr

Abstract. Multimodal problems are omnipresent in the real world: autonomous driving, robotic grasping, scene understanding, etc... Instead of proposing to improve an existing method or algorithm: we will use existing statistical methods to understand the features in already-existing neural networks. More precisely, we demonstrate that a fusion method relying on Canonical Correlation Analysis on features extracted from Deep Neural Networks using different sensors is equivalent to looking at the output of the networks themselves.

Keywords: Multimodal Sensors · Deep Learning · Transport Mode Detection · Inertial sensors · Canonical Correlation Analysis

1 Introduction

Picture a rural scenery: in the countryside, the wind blows through a batch of trees. One can imagine hearing the sound of the wind in the leaves, seeing the branches bend to the gusts of wind, or even feeling the cold air on their skin. All of these stimuli are linked to a single event. Our world is multi-modal: at all times, any event can be captured using a broad diversity of channels. Many real-life problems rely on using multiple modalities: vision and LIDAR sensors for autonomous driving, visual and haptic feedback for robotic grasping, humans even use multiple modalities to understand each other, reading on the lips of their interlocutors.

In the Machine Learning community, a great deal of literature exists to leverage multiple sensors. Some publications use problem-specific solutions, but some approaches are generic: one can, for instance, give the information from all sensors to a single neural network. Or, one can choose to create one network per sensor, to train them to solve the problem the best they can, and to merge the predictions afterwards. In particular, Ahmad *et al.* and Imran *et al.* ([2,5] respectively) performed the fusion using a statistical tool named Canonical Correlation Analysis (CCA), in order to find correlations within two sets of features produced by neural networks trained separately. Their goal was to create a new common representation from all sensors for a gesture recognition problem.

The CCA operation has been used in multiple publications to understand deep neural networks working on a single-modality problem, [12,11,8]. In particular, Roeder *et al.* [13] demonstrated that several architectures, using the same input data, are approximately equal up to a linear transformation. This impressive result was soon followed by McNeely-White *et al.* [10], who showed a similar result for networks working on face recognition.

The present work extends this claim and helps to understand the similarity between the feature neural networks learnt from different sensors. More precisely, we show that the most correlated components between the features from different *sensors* are equal to the class components, *i.e.*, the vectors forming the column of the weight matrix from the classification layer. The most short-term consequence is that the fusion method introduced in [2,5] is equivalent to an average of predictions. To sum up, our contributions are the following:

- we demonstrate that the CCA recomputes the information from the classification layer of the network
- we apply this reasoning to show the fusion methods introduced in [2,5] is identical to a mere average of class logits.

We want to emphasize that we use existing methods and algorithms to reach a new conclusion, which is to show that the use of CCA for data fusion can be replaced by a much less complex equivalent. The rest of this work is organized as follows: section 2 introduces some notations, reviews the Canonical Correlation Analysis, and explains some fundamental concepts to understand our work. Then, we show how the present work is novel compared to the rest of the literature in section 3. Finally, section 4 explains the experiments we led and analyzes the results.

2 Problem position

2.1 Deep Feature Extraction

Let us consider two networks, either two initializations of networks using the same sensor, or networks using different sensors. The most common way to extract features from a network is to record the hidden features right before the last layer (the classification layer), as fig. 1 illustrates. We name these feature matrices X_1 and X_2. An important point to note is that these features are computed from the same samples: if the i^{th} line of X_1 is recorded using an accelerometer segment recorded at a given date, the i^{th} line of X_2 must be computed from data (for instance, magnetometer data) recorded at the same exact moment as the accelerometer segment. The sensors may differ, but the intrinsic samples (and their order in the feature matrices $X_{1,2}$) must correspond.

We call $W_i \in \mathbb{R}^{n_c \times n_i}$ (where n_c is the number of classes and n_i is the number of features from each feature matrix) the *class components*, that is, the column vectors of the weights of the last fully-connected layer (the middle layer in fig. 1). As with every other matrix multiplication, one can understand the classification

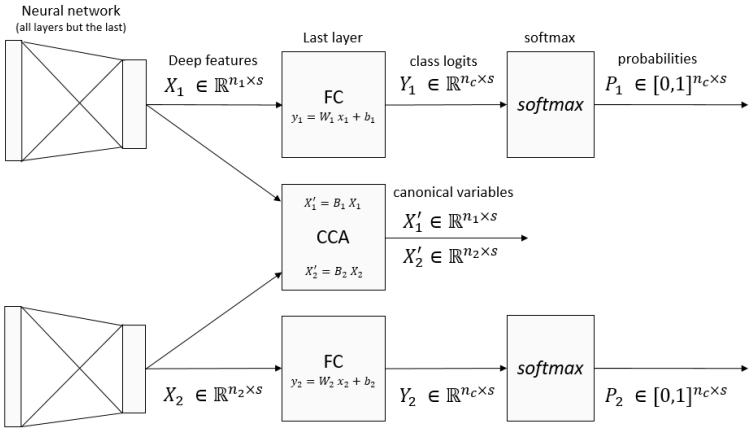


Fig. 1. The extraction of deep features. n_c is the number of classes, n_i is the number of features from each feature matrix, and s is the number of samples

process $x \to W_i.x$ (we omit the bias) as a series of scalar products with the n_c column vectors of W_i: for each class c, the scalar product between each feature vector x and the c^{th} column of W_i gives the logit of class c, a real number giving the likelihood for the sample to belong in class c (the higher the number, the higher the chances that the sample belongs in the class). These logits are then fed into the softmax operation, in order to obtain a series of probabilities (n_c numbers between 0 and 1 that sum to 1).

2.2 Canonical Correlation Analysis

Canonical Correlation Analysis is a statistical tool that takes two feature matrices X_1, X_2, and returns a series of linear combinations of each of these features $X'1 = B_1.X_1$, $X_2' = B_2.X_2$ (where B_1, B_2 are basis change matrices). These new features are defined recursively: the first column of X_1' and the first column of X_2' (the first *canonical variables*) are computed such that the correlation between them is maximized. Then, the second columns of these matrices maximize the correlation between each other while being decorrelated to the previously computed (the correlation between the first and the second columns of X_1' is zero). The subsequent columns are constructed the same way, by maximising the mutual correlation between matrices X_1' and X_2', while being decorrelated to previously computed components. Note that this requires the matrices X_i to be full-rank so that we have enough components. In practice, we use PCA to obtain full-rank feature matrices (we remove the components that account for less than 0.01 % of the cumulative variance).

Similarly to the class components, the *canonical components* are the column vectors of the B_i, and we will compare the first of them to the class components. There are as many canonical components as there are input features in the

feature matrix $X_{1,2}$, but we are only interested in the *first* n_c components. They correspond to the n_c *most* correlated components one can find in the features. We want to show there is a linear relationship between these first n_c canonical components and the n_c class components.

2.3 A simplistic example

Let us consider an unrealistic, but illustrative, example, and let us imagine that the class logits were equal across networks $Y_1 = Y_2$. One should notice that the logits are linear combinations of features $Y_i = W_i.X_i$[3]. This means that one can find linear combinations of features that correlate perfectly with each other. Yet, because of the way the canonical components are computed, these class components will always appear first among the canonical components. In practice, the logits are not equal, but they only need to be correlated enough to each other.

Once this is understood, it is easier to understand the main claim of this work. If we consider two networks, producing the sets of features X_1 and X_2, that succeed fairly at the same classification task, then, the logits Y_1 and Y_2 produced by those networks will be correlated. The last layer of the networks is linear, in other words, the class components can be found among both feature vectors X_1 and X_2 with a simple linear transformation ($Y_i = W_i.X_i$). This means that if one applies CCA to the couple of feature matrices (X_1, X_2), one can find the class components W_i among the first components of B_i.

In particular, this means computing the sum of the canonical variables ($X'_1 + X'_2$, as [2,5] do) is equivalent to summing the logits $Y_1 + Y_2$. Figure 2 illustrates how the equality of the canonical components and the class components make the CCA fusion equivalent to a sum of the logits.

2.4 Extensions

Section 4.2 will detail the experiments we lead to demonstrate the correspondence between CCA components and classification components, both in the case of networks using the same sensor, and in the case of networks using different sensors.

One might wonder what are the causes of the phenomenon. If we sum up the previous sections, for CCA to pick up the class components, two conditions must be verified:

- The class logits of two networks must be more correlated than any other component of the feature space.
- We must apply CCA on the features from the last layer (so that there is a linear relationship between the features and the class logits)

[3] in the following sections, we will omit the bias in the equation $Y_i = W_i.X_i + b_i$. As the CCA assumes that $X_{1,2}$ have zero mean, a necessary step prior to the computation of the canonical components is to remove the mean of the features X_i. This is why adding the constant bias b_i does not change anything to the reasoning

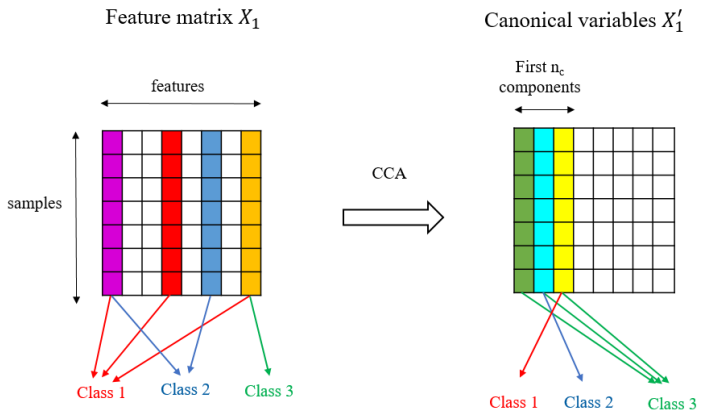


Fig. 2. The principle of the equality between class components and the first canonical components on a three-class problem. The colours in the different feature matrices denote the different information about the three classes. The feature vectors will undergo a matrix multiplication (denoted by the arrows under the left matrix); and the rows of the matrix the features are multiplied by are the class components.

The first condition seems to be verified in practice. For instance, [9] studied the prediction similarity of networks working on the same problem, and found that they are much more similar to each other than what their accuracies could lead to believe. As for the second condition, some publications work with CCA in other layers than the last [12,11] to explore the behaviour of neural networks. However, the verification of this second condition (especially finding an equivalent to the classification components in the earlier layers), is out of scope for this work.

3 Related Works

3.1 CCA as a fusion method

Two works [2,5] used the Canonical Correlation in a multimodal setting: in a problem with several sensors (each of them being able to bring some information about the problem), they both considered the following process: first, they trained one neural network per sensor. Then, they extracted the hidden representations from the last layer of each network. They computed the canonical variables (X'_1 and X'_2 with the notations from the previous section), then summed these components ($X'_1 + X'_2$), before using a Machine Learning algorithm (SVM or KELM) to guess the final prediction from the result of the addition.

Table 7 from [5] shows the results are not very different from averaging of the output *probabilities* of each network. One of our contributions is to show that the CCA operation isolated the class logits in the first components of X'_1 and

X'_2 and that classifying the sum $X'_1 + X'_2$ is roughly equivalent to summing the output *logits* of each network, class by class.

3.2 Similarity of different neural networks

The similarity between two neural networks is a well-studied subject. The closest publications to our work are the ones from Roeder *et al.* [13]. In 2020, they discovered that the representations learnt by different architectures working using the same input data are equal up to a linear transformation; for a broad diversity of tasks including classification. Since this publication, other works, such as [10], expanded the claim to a broad diversity of monomodal architectures and brought new experiments validating this information.

To uncover the similarity between deep models, others looked more at the predictions of networks. For instance, [3] studied the order in which different models learn to classify each sample, while [9] demonstrated that two neural networks classify the samples the same way.

However, all these studies work on monomodal problems, that is, problems with a single input (in most cases, image classification). In the other hand, we provide an example of the similarity between representations learnt with the same architectures, using different *sensors*. The implications are important: this means the networks learnt to exploit the information that remains common between sensors.

3.3 SVCCA and improvements

Several publications worked to improve the computation of the similarity between two bases of features. The first one is SVCCA [12], a famous publication that popularized the use of CCA to measure the similarity between two networks. The idea is to use PCA (Singular Vector decomposition, hence the SV in the abbreviation) on each of the feature matrices to remove low-variance components (which are assumed to be noise), before applying CCA on the reduced feature matrices. We too apply PCA, but only to remove the components with negligible variance (we keep 99.99% of the variance). That is, we keep the 'noisy' low-variance components. To compare, the authors from [12] keep only 99% of the variance, that is, they remove 100 times more variance than us. This means the results we draw are more robust. Morcos *et al.* [11] also noticed the components found by the classic CCA could be noisy. When measuring the proximity between two sets of features X_1 and X_2, they still compute the CCA components $X'_{1,2}$, but instead of using the average of the correlations between X'_1 and X'_2, they choose one of the feature sets (let us say, X_1), and weight the correlation proportionally to the variance which is kept by each CCA component (*i. e.*, the variance of each component of X'_1 divided by the total variance of X_1). This method, named Projection-Weighted CCA (PWCCA), is better at rejecting noise than SVCCA.

Finally, Kornblith *et al.* [8] extended upon this approach, by dividing the correlation by the relative variance of both bases (X'_1 and X'_2). They named their

method CKA (Centered Kernel Alignment) because they use the kernel trick to find better alignments than mere linear combinations.

As [8] states, these two methods are closely related. PWCCA [11] consists in re-weighting the CCA components by the variance of one base, while linear CKA consists in re-weighing the components by using both variances (relatively to the variance of the original sets $X_{1,2}$). To summarize, one can see PWCCA and CKA as different mixtures between PCA and CCA. One could wonder if the conclusions we drew here also apply in the case of PWCCA and CKA. We argue that this is the case, for the following reason: Kamoi *et al.* [7] showed that when a network deals with inliers (non-outliers), the components with the highest variance among the features are approximately equal to a combination of the class logits. This explains the high results of the 'PCA' curve in section 4.2. We showed that the most correlated components are very similar to logits. As a consequence, we expect re-weighting the importance of the CCA components by the amount of variance accounted for by each component to enforce even further the proximity between class logits and most important components. However, this paper focuses on regular CCA, which means that the experiments extending our conclusions to kernel-CCA or CKA are out of scope.

4 Experiments

In this section, we will first reproduce the results from [12], in order to illustrate our experimental protocol (section 4.2). Then, we will repeat this experiment mixing data from different sensors to show our main claim in section 4.2.

4.1 Datasets

CIFAR10 We use the famous ResNet-56 network [4] on the CIFAR-10 Dataset. This is a Computer Vision classification problem, where the model has to classify low-resolution (32×32) images into ten classes. The dataset contains 50,000 training samples and 10,000 validation samples. We trained the network hyper-parameters and architecture as the original publication [4] thanks to the code from [1].

The dataset has only one sensor (the RGB images), but we work with different initializations of networks that use the same modality. We use this dataset to provide a comparison with the rest of the literature on CCA with deep features, as most works chose to include a ResNet trained on CIFAR-10 [12,11,8].

SHL 2018 dataset The Sussex-Huawei Locomotion 2018 dataset is a Transport Mode detection problem. Organizers asked three participants to record the sensor values from several smartphones while travelling using different modes (walking, running, driving, *etc.*). Then, the data is published, and a yearly challenge is organized to get a precise evaluation of the state of the art. The 2018 dataset is the first version of the challenge: only the data from a single user and a single

sensors	Accelerometer, Gravity, Linear Acceleration, Gyrometer, Magnetometer, Orientation quaternion, barometric Pressure
classes	Still, Walk, Run, Bike, Car, Bus, Train, Subway
segment duration	$60s$
sampling frequency	$100Hz$
training samples	$13,000$
validation samples	$3,310$

Table 1. An overview of the SHL 2018 dataset

smartphone (the one in the hand) is available for classification. The organizers released 16,310 annotated samples for training and validation.

The dataset includes seven sensors (accelerometer, gravity, linear acceleration, gyrometer, magnetometer, orientation vector, barometric pressure), most of them having several axes (x,y,z). We will study three signals among them: the y axis of the gyrometer (`Gyr_y`), the norm of the acceleration (`Acc_norm`, as in [6]), and the norm of the magnetometer (`Mag_norm`). The accelerometer and gyrometer encode similar information (they record the inertial dynamics of the sensor) and are most useful when detecting walk, run, or bike segments. On the other hand, the norm of the magnetometer mostly changes when the sensor is close to a strong magnetic field: far from any ferromagnetic object, its values stays close to $40\mu T$ (the value of the Earth's magnetic field). But this sensor can go up to $200\mu T$ when a strong magnetic field is present (for instance, when the sensor is close to a ferromagnetic object or even an electrical engine). This is why we think this sensor will be best to detect the train or subway classes from the rest. To summarize, the accelerometer and gyrometer are expected to be similar to each other, while these sensors encode different information than the magnetometer. This is intended to represent different relatedness between sensors.

We use the same approach as in [6], each signal is first converted into a two-dimensional spectrogram (a time-frequency diagram) using short term Fourier transform. The frequency axis of the spectrogram is rescaled using a logarithmic scale, in order to give more resolution to the lower frequencies. This method aims to give better resolution to the $2-3Hz$ frequency bands (which are the most useful to distinguish the Walk, Run, and Bike segments), while still keeping the highest frequencies available. See [6] for more information and illustrations. For each sensor, we obtain a $48 \times 48 \times 1$ spectrogram, that is fed into a CNN which architecture is simple: three convolutional layers (with 16, 32, and 64 filters), and two fully-connected layers (with 128 hidden features and 8 output features). See [6] for details about hyperparameter or training process.

To illustrate, on three random initializations, the average validation F1-score of each of these individual sensors is 89 % for the accelerometer, 80 % for the gyrometer, and 67 % for the magnetometer.

In both experiments, we will use a train set to train the models, extract the features, compute the base change with CCA, and, when applicable, retrain the

models. The validation sets only go through trained models and already-computed base changes, before being used to display a result. We want to emphasize that when dealing with multimodal sensors, each network was only trained on a single modality: the network using the accelerometer never saw the gyrometer or magnetometer data, and so on.

4.2 Studying component similarity with subspace projection

In this section, we will reproduce and extend the experiments from [12] (Figure 2 from this work). As fig. 3 illustrates, we start from a trained network, we extract the hidden features form the last layer, then we project on a subspace of inferior dimension, before re-injecting the features in the network to measure the performance. If the performance is intact, it means that the n_c class components are unaffected by the projection. In other words, it means the class components already belong in the image of the projection. In particular, when the dimension of the image of the projection is $n_s = n_c$, and if the performance is unchanged, it means that the n_c class components belong in the subspace spanned by the n_s most correlated components, which implies the existence of a linear relationship between the families (as the canonical components and class components are both linearly independent families of vectors).

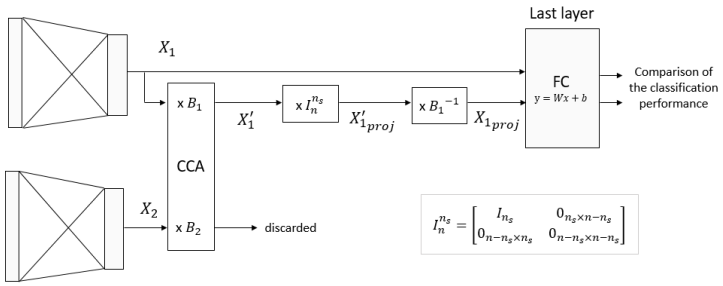


Fig. 3. The principle of the subspace projection experiment: $P_1 = B_1^{-1}.I_n^{n_s}.B_1$ projects X_1 onto a linear space with dimension n_s.

Note that when we use all features, we project on the original space, *i.e.*, we leave the data unchanged. The difference between the end of the curves (performance on pristine data) and the rest (altered data) will indicate the proximity between the considered subspace and class components.

The dimension and the way of choosing the subspace will vary: as in [12], we consider choosing the n most correlated components found with CCA (`CCA_highest`), the n features with the highest activation in absolute value (`max_activation`), and n features chosen randomly (`random_selection`). In order to provide comparisons, we add four reductions methods that are not included in [12]:

- *random orthogonal projection* (`random_projection`). Comparing the random selection of n components versus the projection on n components shows that the canonical basis does not play a particular role (*i.e.* selecting the values of n features is not particularly meaningful).
- *PCA*: Kamoi *et al.* [7] showed that the n_c components with the most variance are the components that will be used for classification. We project the features on the components with most variance to validate their findings.
- *least correlated components* (`CCA_lowest`): if the most correlated components are the class components, the components with lowest correlation should not include any relevant information for the problem.
- *CCA with random components* (`CCA_random`): one may argue that the CCA curve is above the others in [12] because CCA allows to create decorrelated components, which would mean that its components are less redundant than random directions. If this was the case, selecting random CCA components would be better than selecting components with a random projection.

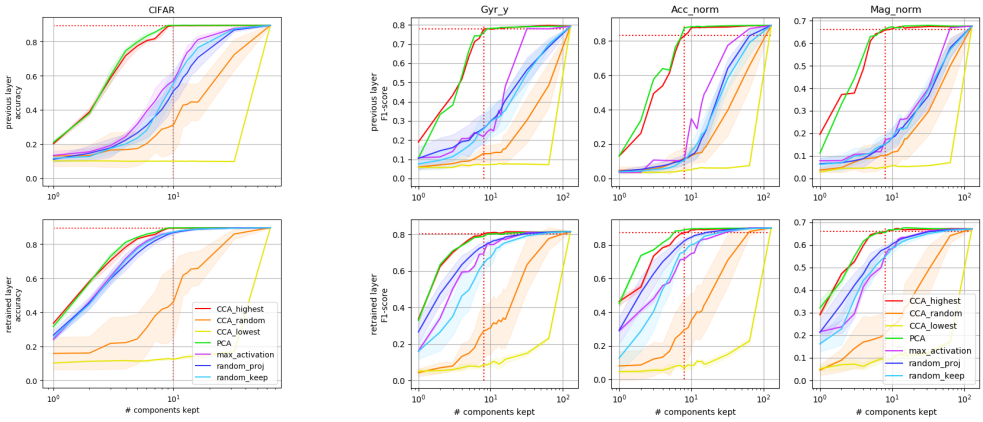
To save time, we do not consider all the possible number of components: because we want a high resolution around n_c, we only considered the $2 * n_c$ first components (where n_c is the number of classes, 8 for SHL and 10 for CIFAR), and, after that, the number of components which are powers of 2 $(16, 32, ...)$, up to the maximal number of components (128 for SHL, 64 for CIFAR)

In addition to this, after measuring the performance of the layer when using projected features, we also try retraining the classification layer on a projected version of the validation set, with the same hyperparameters as the initial training of the network. The goal of this retraining is to illustrate the difference between the components a network actually uses for classification and the components that carry some information about the problem. If the performance of the retrained layer is low, this means we can be sure that the projection removed *all* useful information. If only the performance of the original layer is low, this only means that we got rid of the information that was used by the network.

Note that the CCA operation requires two databases. When we use CCA, we use a second matrix of features X_2, but only to compute X_1' (we discard X_2'). In the next section, this second network is another initialization of a network working with the same sensor, while the section after that shows experiments made with two networks using different sensors.

Similarity between identical sensors Figure 4 shows the result of this experiment. We can draw several conclusions from it:

- The performance of the projection on the n_s highest variance components ('pca', green curve) is maximal for $n_s = n_c$: this verifies the findings of Kamoi *et al.* [7], the n_c components with highest variance are the class components.
- Similarly to Figure 2 from [12], the most correlated components are more useful for the classification problem than a random choice of components from the canonical basis.



(a) (b)

Fig. 4. The performance of the networks after projecting their features on subspaces with varying dimensions, on the CIFAR (a) and SHL (b) validation sets. The top row indicates the validation performance of the network as is, while the bottom row indicates the performance when retraining the classification layer on a projected training set. For each curve, the experiment was repeated 5 times, and the standard deviation is given by the width of the curve (which is sometimes too small to see). The dotted line highlights the performance with the n_c most correlated components. Best view in colour.

- The red curve (performance of the components with the highest correlation) is almost at its maximum for n_c components even before retraining, there is almost nothing to gain after n_c components. This validates our original claim these components correspond to the subspace used by the classification layer.
- The yellow curve (the components with lowest correlations), is under all the others. Before retraining, the performance of a projection on the n components with the lowest correlation is minimal, even when we select half the components. After retraining, the performance of these components is still well under the performance of random components: selecting the least correlated components effectively removed most of the classification information.
- The orange curve (random CCA components) is lower than the random choice of components (blue curves). This means that the performance of the components with the highest correlation is not due to an efficient encoding. Additionally, the standard deviation of this curve is unusually high: as the CCA operation isolates the classification components from the rest, selecting some of its components at random creates extremes situations: either a classification component is selected, or it is not. The standard deviations of the other random methods are not as high because the random choices allow to span partly the classification components.
- Before retraining, the two blue curves are equivalent, this indicates that the canonical components do not play any specific role in regards to classification. After retraining, the dark blue curve (random projection) is higher than light blue (random selection of canonical components). We hypothesize that the canonical components carry some redundancy between them because of the dropout we used to train our networks, and that re-training the network allows it to stop expecting this redundancy in the features it sees.

Similarity between two different sensors We now lead experiments to verify our main contribution: the fact that the most correlated features are equal to classification components, even when the correlation is computed across sensors. This time, when computing CCA, we use features from a network using different sensors. In this section, we do not include any of the other dimensionality reduction methods (PCA, random projection and selection of components, maximal activation components) because those methods work with only one database: the results would be copies of the curves presented in fig. 4.

Figure 5 shows that the performance is maximal when the number of components is close to 10 or 20, approximately. However, contrary to fig. 4b, the performance is off by a few points when the number of selected components is equal to 8, the number of classes. This means that the equality between most correlated components and classification vectors is less strong than in the previous case when the CCA was computed from the same sensor. Still, the performance with only 8 components is high enough for us to conclude that the components computed with CCA overlap significantly with the classification components.

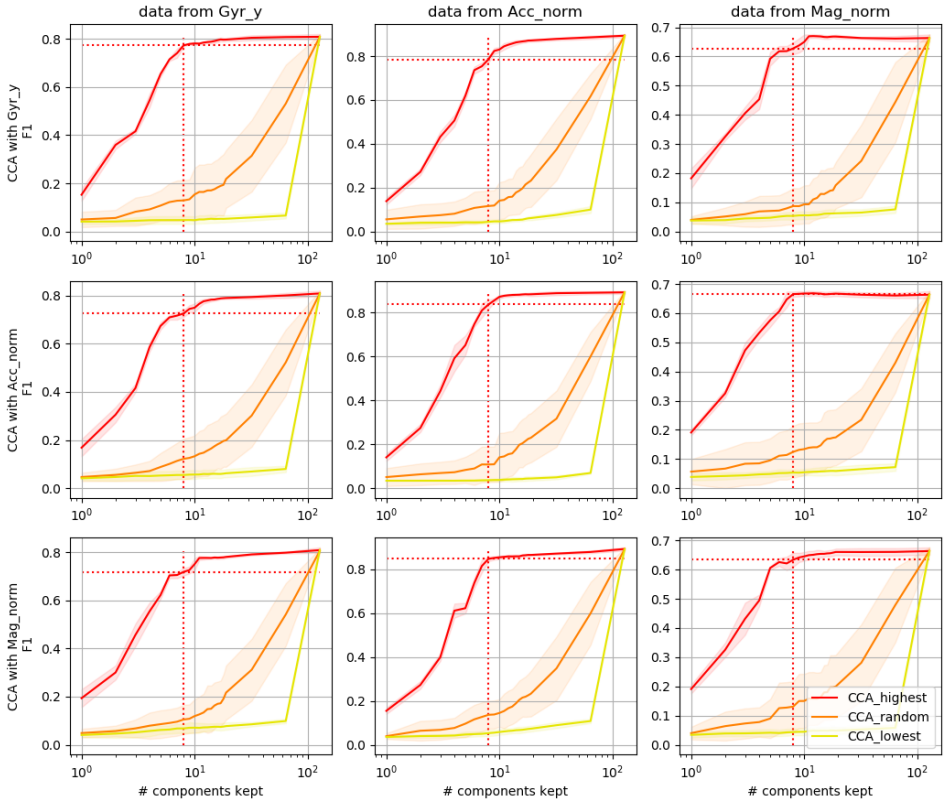


Fig. 5. The classification performance of classification layer using features projected on a subspace with varying dimension, when the CCA is computed thanks to data from another sensor. As in fig. 4, one can see that the performance with the n_c most correlated components is close to the performance with all components. The graphs in the diagonal were generated using the same protocol as the first row of graphs in fig. 4b. The dotted line highlights the performance with the n_c most correlated components. Best view in colour.

5 Conclusion

We began by demonstrating the experiments from previous publications: the findings from Kamoi *et al.* [7], who showed that the components with most variance are the classification components, and Roeder *et al.* [13], who showed that CCA found the classification components when it is applied to features from monomodal networks using the same dataset. In a later section, we showed the same result held when applied to features learnt from different sensors, indicating that the networks exploit information that can be found across multiple sensors. The exact nature of this information, however, is yet to be found.

In addition to showing that the fusion method from [2,5] is unnecessarily complex, these results have strong implications for multimodal learning: in this situation, it may be unnecessary to add too many sensors, for neural networks would compute similar information.

Future work might include finding exactly the nature of the common information which is present in all sensors' signals and exploited by neural networks. Or, we could try to explain a paradox about the similarity we measured: the networks using the accelerometer, gyrometer, and magnetometer have average F1-scores of about 90%, 80%, and 67% (respectively). How can couples of features that are so similar have such different performance levels?

References

1. 3.2.2 ResNet_cifar10 - PyTorch Tutorial, https://pytorch-tutorial.readthedocs.io/en/latest/tutorial/chapter03_intermediate/3_2_2_cnn_resnet_cifar10/
2. Ahmad, Z., Khan, N.: Human Action Recognition Using Deep Multilevel Multimodal (M^2) Fusion of Depth and Inertial Sensors. IEEE Sensors Journal **20**(3), 1445–1455 (Feb 2020). https://doi.org/10.1109/JSEN.2019.2947446
3. Hacohen, G., Choshen, L., Weinshall, D.: Let's Agree to Agree: Neural Networks Share Classification Order on Real Datasets. In: International Conference on Machine Learning. pp. 3950–3960. PMLR (2020)
4. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016), http://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html
5. Imran, J., Raman, B.: Evaluating fusion of RGB-D and inertial sensors for multimodal human action recognition. Journal of Ambient Intelligence and Humanized Computing **11**(1), 189–208 (Jan 2020). https://doi.org/10.1007/s12652-019-01239-9, https://doi.org/10.1007/s12652-019-01239-9
6. Ito, C., Cao, X., Shuzo, M., Maeda, E.: Application of CNN for Human Activity Recognition with FFT Spectrogram of Acceleration and Gyro Sensors. In: Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18. pp. 1503–1510. ACM Press, Singapore, Singapore (2018). https://doi.org/10.1145/3267305.3267517, http://dl.acm.org/citation.cfm?doid=3267305.3267517

7. Kamoi, R., Kobayashi, K.: Why is the Mahalanobis Distance Effective for Anomaly Detection? arXiv:2003.00402 [cs, stat] (Feb 2020), http://arxiv.org/abs/2003.00402, arXiv: 2003.00402
8. Kornblith, S., Norouzi, M., Lee, H., Hinton, G.: Similarity of Neural Network Representations Revisited. In: International Conference on Machine Learning. pp. 3519–3529. PMLR (May 2019), http://proceedings.mlr.press/v97/kornblith19a.html
9. Mania, H., Miller, J., Schmidt, L., Hardt, M., Recht, B.: Model Similarity Mitigates Test Set Overuse. In: Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F.d., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019), https://proceedings.neurips.cc/paper/2019/file/48237d9f2dea8c74c2a72126cf63d933-Paper.pdf
10. McNeely-White, D., Sattelberg, B., Blanchard, N., Beveridge, R.: Exploring the Interchangeability of CNN Embedding Spaces. arXiv:2010.02323 [cs] (Feb 2021), http://arxiv.org/abs/2010.02323, arXiv: 2010.02323
11. Morcos, A., Raghu, M., Bengio, S.: Insights on representational similarity in neural networks with canonical correlation. Advances in Neural Information Processing Systems **31**, 5727–5736 (2018)
12. Raghu, M., Gilmer, J., Yosinski, J., Sohl-Dickstein, J.: SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 6076–6085. Curran Associates, Inc. (2017)
13. Roeder, G., Metz, L., Kingma, D.P.: On Linear Identifiability of Learned Representations. arXiv:2007.00810 [cs, stat] (Jul 2020), http://arxiv.org/abs/2007.00810, arXiv: 2007.00810